FUJITSU

FUJITSU Software NetCOBOL V11.0

Language Reference

B1WD-3304-03ENZ0(00)
August 2015

Preface

This NetCOBOL Reference Manual covers the rules for writing programs in COBOL (COmmon Business Oriented Language). It contains information on the base language, as well as extensions and restrictions for Fujitsu NetCOBOL and NetCOBOL for the .NET environment.

The reader is assumed to have a basic knowledge of programming.

Organization

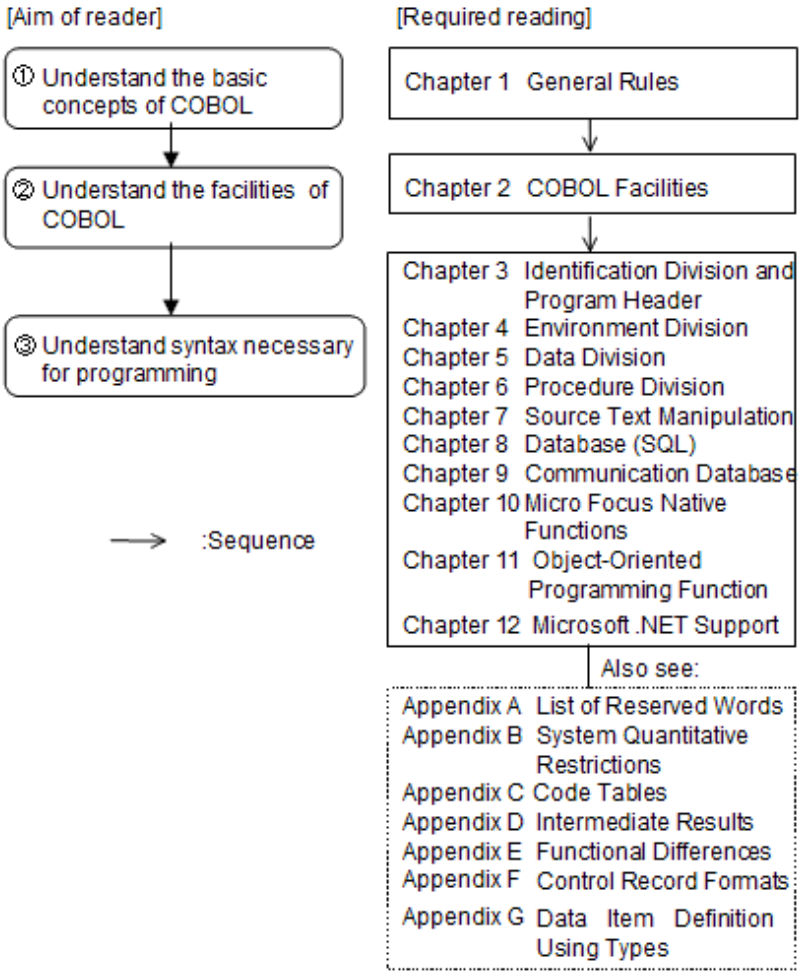
The table below shows how this manual is organized.

Chapter/Appendix	Description
Chapter 1. General Rules	Covers the general rules including the language elements of COBOL, unique reference, writing of literals, and reference format
Chapter 2. COBOL Modules	Lists and explains the facilities of COBOL
Chapter 3. Identification Division and End Program Header	Explains the syntax of the Identification Division and End Program header
Chapter 4. Environment Division	Explains the syntax of the Environment Division
Chapter 5. Data Division	Explains the syntax of the Data Division
Chapter 6. Procedure Division	Explains the syntax of the Procedure Division
Chapter 7. Source Text Manipulation	Explains the syntax of statements used in the Source Text Manipulation function
Chapter 8. Database(SQL)	Explains the syntax of the Database function(SQL)
Chapter 9. Communication Database	Explains the syntax of the Communication Database function
Chapter 10. Micro Focus Native Functions	Explains the syntax of the Micro Focus native functions
Chapter 11. Object-Oriented Programming Function	Explains the syntax of the object-oriented programming function.
Chapter 12. Microsoft .NET Support	Describes the extensions added to support the .NET Framework
Appendix A. List of Reserved Words	Lists the reserved words in COBOL
Appendix B. System Quantitative Restrictions	Lists the quantitative restrictions of the COBOL Runtime System
Appendix C. Code Tables	Lists the character sets and their internal representations
Appendix D. Intermediate Results	Explains the attributes and accuracy of intermediate results
Appendix E. Functional Differences	Lists the functional differences for each operating system
Appendix F. Control Record Formats	Explains the formats of the control records
Appendix G. Data Item Definition Using Types	Explains the usage of types.

How to Use this Manual

Users who wish to gain an understanding of the basic concepts of COBOL and its facilities should first read Chapters 1 and 2 in that order. Users who already have an understanding of COBOL may skip the first two chapters and read any of the other chapters or appendices as required.

The following diagram shows the sequence in which the chapters and appendices of this reference manual should be read.

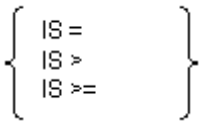


Symbols Used in Format Diagrams

In each chapter, Format shows how to write COBOL language elements, such as statements and clauses. The words shown in Format must be written in the sequence shown unless otherwise specified in the syntax rules or general rules.

Symbol	Meaning	Example
Character-string in uppercase letters	COBOL reserved word. These character-strings must be written exactly as they appear in Format.	**Example** VALUE IS constant-1 **Meaning** VALUE cannot be omitted. IS can be omitted. Write any constant conforming to the syntax rules in constant-1. **For example** VALUE IS "XXXX" VALUE 12345
_____ (underline)	Underlining indicates that the character-string is a key word. A key word cannot be omitted. A character-string that is not underlined can be omitted.	
Character-string in lowercase letters	Character-string in lowercase letters indicates the classifications of user-defined word, literal, PICTURE clause character-strings, and comment entries. Any character-string can be written in this portion provided that it conforms to the syntax rules.	
[ ] (brackets)	Brackets indicate that one of the values given in the parentheses can be selected, or the contents of the brackets can be omitted.	**Example 1**

Symbol	Meaning	Example
		[data-name-1 FILLER] **Meaning** Write data-name-1 or FILLER, or omit this value completely. **Example 2** [ON SIZE ERROR unconditional-statement-1] **Meaning** This value can be omitted completely.
{ } (braces)	Braces indicate that one of the values given in the braces can be selected or the value can be omitted.	**Example 1** {identifier-1 | literal-1} **Example 2** { identifier-1 literal-1 } **Meaning** Write either identifier-1 or literal-1.
{| |} (choice indicators)	These indicate that at least one of the values given in the choice indicators can be written. However, each character-string must be unique.	**Example** { | COMMON | } { | INITIAL | } **Meaning** Any of the following can be written: - COMMON - INITIAL - COMMON INITIAL - INITIAL COMMON
... (occurrence symbol)	Indicates that the portion immediately preceding the occurrence symbol (the portion enclosed in brackets or parentheses) can be repeated.	**Example** { identifier-1 literal-1 } ... **Meaning** Identifier-1 or literal-1 can be repeated.
. (period)	Any period shown in Format must be written in the same position.	**Example** `WORKING-STORAGE SECTION.` **Meaning** The period after SECTION must be written.

Symbol	Meaning	Example
Special characters such as +, -, >, <, =, >=, <=, ->	Special characters are key words. They are not underlined, but must not be omitted.	**Example**  **Meaning** The symbol =, > or >= must be written.

F

Shading

The shaded portions in the documentation indicate that they describe extended functions or functions specific to NetCOBOL. If the heading of a section or an item is shaded, the entire description under that heading is an extended function of NetCOBOL.

Syntax Rules and General Rules

The explanation of COBOL language elements such as statements and clauses are arranged into Format, syntax rules, and general rules.

Format shows the arrangement of elements making up a statement or clause.

Syntax rules explain the arrangement of the elements in Format and restrictions on their arrangement.

General rules explain the results of execution and compilation where a statement or clause was written. They also explain the meaning of elements in Format and the relationship between the elements.

Syntax rules or general rules are omitted when there are no rules relating to the elements in Format.

Obsolete Elements

Elements marked "Obsolete elements" in the text are given in the 1985 issue of the ANSI COBOL standard but may not be included in the next edition. Fujitsu recommends that obsolete elements not be used when creating new programs.

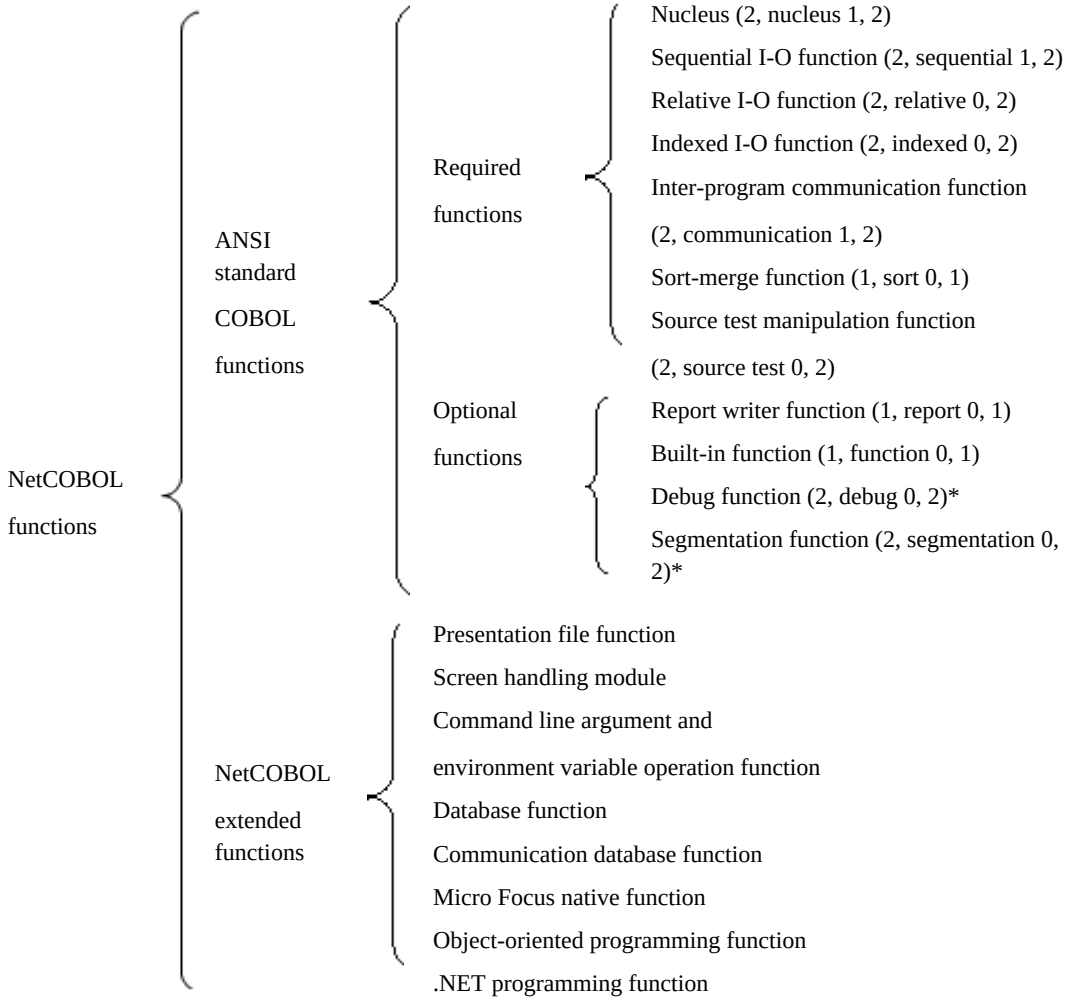
Related Documentation

The documentation set for NetCOBOL includes "NetCOBOL User's Guide" in addition to this document. Refer to "NetCOBOL User's Guide" for creating, compiling, executing, and debugging programs.

Refer to "NetCOBOL User's Guide" for the details of the other systems.

Scope of NetCOBOL Functions

The NetCOBOL functions consist of ANSI standard COBOL functions (ANSI Standard COBOL X3.23 1985, ISO-1989-1985) and NetCOBOL extended functions. The NetCOBOL functions are as follows:



*: The debug function and segmentation functions are treated as comments when the program is run.

Note

In the above, standard mnemonic symbols for each ANSI standard COBOL function are shown in parentheses. From left to right the symbols show the position of the function in the hierarchy, the function abbreviation, and the maximum and minimum levels of the function to which the level belongs.

System-specific Functions

Some parts of the COBOL common syntax described in this manual depend on system functions, and differ among systems.

Such parts are indicated by the following system names:

Indicator	Corresponding system	Corresponding product
[DS] [UXP/DS]	UXP/DS COBOL85	V20L11
[HP]	HP-UX COBOL85	V20L11
[Solaris]	Oracle Solaris 10	NetCOBOL V10
[Linux]	Red Hat(R) Enterprise Linux(R) 5(for x86) Red Hat(R) Enterprise Linux(R) 5(for Intel64) 32bit	NetCOBOL (32bit) V7.3
[LinuxIPF]	Red Hat(R) Enterprise Linux(R) 5(for Intel Itanium)	NetCOBOL V9.0

Indicator	Corresponding system	Corresponding product
[Linux64]	Red Hat(R) Enterprise Linux(R) 6(for Intel64) 64bit Red Hat(R) Enterprise Linux(R) 7(for Intel64) 64bit	NetCOBOL V11
[Win16]	Windows 95 Windows 3.1	COBOL V20L11
[Win32]	Windows Server 2012 R2 Windows Server 2012 Windows Server 2008 R2 Windows 8.1 Windows 8 Windows 7	NetCOBOL (32bit) V11
[Winx64]	Windows Server 2012 R2 Windows Server 2012 Windows Server 2008 R2 Windows 8.1(x64) Windows 8(x64) Windows 7(x64)	NetCOBOL (64bit) V11
[.NET]	Windows Server 2012 R2 Windows Server 2012 Windows Server 2008 R2 Windows 8.1 Windows 8 Windows 7	NetCOBOL for .NET V6

Refer to Appendix E "Functional Differences" for a list of functional differences.

Product Names

Product Name	Abbreviation
Microsoft® Windows Server® 2012 R2 Datacenter Microsoft® Windows Server® 2012 R2 Standard Microsoft® Windows Server® 2012 R2 Essentials Microsoft® Windows Server® 2012 R2 Foundation	Windows Server 2012 R2
Microsoft® Windows Server® 2012 Datacenter Microsoft® Windows Server® 2012 Standard Microsoft® Windows Server® 2012 Essentials Microsoft® Windows Server® 2012 Foundation	Windows Server 2012
Microsoft® Windows Server® 2008 R2 Foundation Microsoft® Windows Server® 2008 R2 Standard Microsoft® Windows Server® 2008 R2 Enterprise Microsoft® Windows Server® 2008 R2 Datacenter	Windows Server 2008 R2
Windows® 8.1	Windows 8.1

Product Name	Abbreviation
Windows® 8.1 Pro Windows® 8.1 Enterprise	or Windows 8.1 (x64)
Windows® 8 Windows® 8 Pro Windows® 8 Enterprise	Windows 8 or Windows 8 (x64)
Windows® 7 Home Premium Windows® 7 Professional Windows® 7 Enterprise Windows® 7 Ultimate	Windows 7 or Windows 7 (x64)
Oracle Solaris	Solaris

Trademarks

- NetCOBOL is a trademark or registered trademark of Fujitsu Limited or its subsidiaries in the United States or other countries or in both.
- HP and HP-UX are trademarks of Hewlett-Packard Inc., U.S.A.
- Micro Focus is a trademark of Micro Focus International Limited
- Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. Oracle Solaris might be described as Solaris, Solaris Operating System, or Solaris OS.
- Windows, Windows Server, MSDN, Visual Studio, and .NET are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.
- Linux is a registered trademark of Mr. Linus Torvalds in the United States and other countries.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel and Itanium are registered trademarks of Intel Corporation.
- Other brand and product names are trademarks or registered trademarks of their respective owners.

NetCOBOL is copyrighted by Fujitsu Limited with all rights reserved. As a component of that program NetCOBOL uses libXpm developed by Groupe BULL and licensed to Fujitsu Limited under the requirement to reflect the following permission only as it pertains to the libXpm.

Copyright 1989-94 GROUPE BULL

Permission is hereby granted, free of charge, to any person obtaining a copy of the libXpm and associated documentation files (the "libXpm software"), to deal in the libXpm software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the libXpm Software, and to permit persons to whom the libXpm Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE LIBXPM SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRIGEMENT. IN NO EVENT SHALL GROUPE BULL BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE LIBXPM SOFTWARE OR THE USE OR OTHER DEALINGS IN THE LIBXPM SOFTWARE.

Except as contained in this notice, the name of GROUPE BULL shall not be used in advertising or otherwise to promote the sale, use or other dealings in this libXpm software without prior written authorization from GROUPE BULL.

Export Regulation

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Acknowledgment

COBOL language specifications are based on the original specifications developed by the Conference on Data Systems Languages (CODASYL), and specifications in this manual are also derived from these specifications. The chapters listed below were included on request from CODASYL.

COBOL is not owned by any particular company, organization, or group; it has been designed for general use in industry. CODASYL does not guarantee and bears no responsibility in relation to the programming method, the accuracy of language, and functions.

The following copyright owners permitted partial use of the following documents when the original specifications were put together. This permission also extends to the use of the original specifications in other COBOL specifications.

- FLOW-MATIC (trademark of Sperryrand Inc.), "Programming for the Univac I and II, Data Automation Systems", copyright Sperryrand Inc. 1958, 1959
- IBM Commercial Translator, library number F28-8013, copyright IBM Inc. 1959
- FACT, library number 27A5260-2760, copyright Minneapolis Honeywell Inc. 1960

The contents of this manual may be revised without prior notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Fujitsu Limited.

August 2015

Copyright 1996-2015 FUJITSU LIMITED

Contents

Chapter 1 General Rules....1
1.1 Characters and Character Sets....1
1.2 Basic Overview of the Language....2
1.2.1 Separator....2
1.2.2 COBOL Word....4
1.2.2.1 User-defined Words....5
1.2.2.1.1 Rules for Describing User-defined Words....6
1.2.2.1.2 Application of User-defined Words....7
1.2.2.2 System Name....8
1.2.2.3 Reserved Words....9
1.2.2.4 FUNCTION-NAME....11
1.2.3 Literal....11
1.2.3.1 Numeric Literal....11
1.2.3.2 Nonnumeric Literal....12
1.2.3.3 Hexadecimal Nonnumeric Literal....13
1.2.3.4 National Nonnumeric Literal....14
1.2.3.5 Boolean Literal....15
1.2.4 Figurative Constant....15
1.2.5 Concatenation Expression....17
1.2.6 Literals for Special Applications....17
1.2.7 Picture Character-string....18
1.2.8 Comment Entry....18
1.3 Concept of Data Description....18
1.3.1 Concept of Levels....18
1.3.2 Concept of Class....18
1.3.3 Concept of Types....19
1.3.3.1 Weakly-Typed Item....19
1.3.3.2 Strongly-Typed Item....20
1.3.4 Standard Alignment Rule....21
1.3.5 Adjustment of Data Boundaries....21
1.3.5.1 Slack Byte....21
1.3.5.2 Slack Bit....23
1.4 Uniqueness of Reference....25
1.4.1 Qualification....25
1.4.2 Subscripting....28
1.4.3 Reference Modification....29
1.4.4 Pointer....31
1.4.5 Identifier....32
1.4.6 Condition-name Uniqueness of Reference....32
1.4.7 Function-identifier....33
1.5 Reference Format....33
1.5.1 Configuration of a Source Code Line....34
1.5.2 Reference Format for Area A and Area B....35
1.5.3 Blank Line....35
1.5.4 Comment Line....36
1.5.5 Continuation of Lines....36
1.5.6 Debugging Line....37
1.5.7 In-line Comment....37
1.5.8 Reference Format for Free Format....37
1.5.8.1 Configuration of a Line....37
1.5.8.2 Blank Line....38
1.5.8.3 Comment Lines....38
1.5.8.4 Continuation of Lines....38
1.5.8.5 Debugging Line....38

1.6 Program Configuration ... 38
1.6.1 Returning Result from a Program ... 39
1.7 Operation Mode ... 39

Chapter 2 COBOL Modules ... 41
2.1 Nucleus ... 41
2.1.1 Transcription and Movement of Data ... 43
2.1.2 Arithmetic Operations ... 44
2.1.3 Optional Processing and Branching ... 44
2.1.4 Repetitive Processing ... 45
2.1.5 Table Handling ... 46
2.1.6 Initialization of Data Items ... 49
2.1.7 Handling Character Strings ... 49
2.1.8 Simple Input-Output ... 50
2.1.9 Terminating a Program ... 50
2.1.10 Pointer Handling ... 50
2.1.11 Handling Floating-Point Data Items ... 51
2.2 Input-Output Facility ... 52
2.2.1 File Organization ... 53
2.2.2 File Connector ... 54
2.2.3 Operation of Input-Output Statements ... 55
2.2.4 File Position (Record Pointer) Indicator ... 56
2.2.5 Volume Indicator ... 56
2.2.6 Sharing and Exclusion of Files ... 56
2.2.7 Locking a Record ... 57
2.2.8 I-O Status ... 58
2.2.9 Record Format ... 59
2.2.10 Record Area ... 59
2.2.11 LINAGE-COUNTER Special Register ... 60
2.3 Inter-program Communication Module ... 60
2.3.1 Accessing and Returning to a Program ... 61
2.3.2 Global Name and Local Name ... 61
2.3.3 External Attribute and Internal Attribute ... 62
2.3.4 External Name and Internal Name ... 62
2.3.5 Program Activation ... 63
2.3.6 The Common Attribute ... 63
2.3.7 Initial State of a Program ... 63
2.3.8 Recursive Attribute of the Program ... 64
2.3.9 Passing Parameters to the Called Program ... 64
2.3.10 Scope of Names ... 65
2.3.11 Scope of a Program-name and Secondary Entry Point Names(*) ... 66
2.3.12 PROGRAM-STATUS and RETURN-CODE Special Registers ... 67
2.4 Sort-Merge Module ... 67
2.4.1 Sorting Methods ... 68
2.4.2 Merging Methods ... 68
2.4.3 SORT Input Procedure ... 68
2.4.4 Sort-Merge Output Procedures ... 68
2.4.5 Sort-merge File ... 68
2.4.6 Special Register ... 69
2.4.6.1 SORT-STATUS ... 69
2.4.6.2 SORT-CORE-SIZE ... 69
2.5 Source Text Manipulation Module ... 69
2.6 Presentation File Module ... 70
2.6.1 Destination Type ... 71
2.6.2 Screen Form Descriptor ... 72
2.6.3 File Organization and Access Mode ... 72
2.6.4 Presentation File Input-Output Statements ... 72

2.6.5 I-O Status ... 72
2.6.6 Presentation File Special Registers ... 73
2.7 Intrinsic Function Module ... 74
2.8 Screen Handling Module ... 76
2.8.1 Screen and Screen Item ... 76
2.8.1.1 Screen Data Description Entry ... 76
2.8.1.2 Screen Item ... 76
2.8.2 Input-Output Handling of the Screen ... 77
2.8.3 Screen Input Status ... 77
2.9 Command Line Argument and Environment Variable Modules ... 77
2.9.1 Processing Command Line Arguments ... 78
2.9.1.1 Method for Finding the Number of Command Line Arguments ... 78
2.9.1.2 Method for Obtaining the Value of Individual Command Line Arguments ... 78
2.9.2 Accessing an Environment Variable ... 79
2.9.2.1 Accessing the Value of an Environment Variable ... 79
2.9.2.2 Updating the Value of an Environment Variable ... 79
2.10 Report Writer Module ... 79
2.10.1 Report File ... 81
2.10.2 Report Writer Special Registers ... 81
2.10.2.1 Page Counter ... 81
2.10.2.2 Line-Counter ... 81

Chapter 3 Identification Division and End Program Header ... 82
3.1 Composition of the Identification Division ... 82
3.1.1 Program-Id Paragraph ... 82
3.1.2 DATE-COMPILED Paragraph ... 83
3.2 End Program Header ... 84

Chapter 4 Environment Division ... 86
4.1 Composition of the Environment Division ... 86
4.2 CONFIGURATION SECTION ... 86
4.2.1 SOURCE-COMPUTER Paragraph ... 86
4.2.1.1 WITH DEBUGGING MODE Clause ... 87
4.2.2 OBJECT-COMPUTER Paragraph ... 87
4.2.2.1 MEMORY SIZE Clause ... 87
4.2.2.2 PROGRAM COLLATING SEQUENCE Clause ... 88
4.2.3 SPECIAL-NAMES Paragraph ... 89
4.2.3.1 Function-name-1 Clause ... 89
4.2.3.2 Function-name-2 Clause ... 92
4.2.3.3 Function-name-3 Clause ... 93
4.2.3.4 ALPHABET Clause ... 93
4.2.3.5 CLASS Clause ... 96
4.2.3.6 CRT STATUS Clause ... 97
4.2.3.7 CURRENCY SIGN Clause ... 98
4.2.3.8 CURSOR Clause ... 98
4.2.3.9 DECIMAL-POINT IS COMMA Clause ... 99
4.2.3.10 POSITIONING UNIT Clause ... 99
4.2.3.11 PRINTING MODE Clause ... 99
4.2.3.12 SYMBOLIC CHARACTERS Clause ... 101
4.2.3.13 SYMBOLIC CONSTANT Clause ... 102
4.3 Input-Output Section ... 103
4.3.1 File-Control Paragraph ... 103
4.3.1.1 ACCESS MODE Clause (Sequential, Relative, Indexed, Presentation(*), and Report Writer Files) ... 110
4.3.1.2 ALTERNATE RECORD KEY Clause (Indexed File) ... 112
4.3.1.3 ASSIGN Clause (Sequential File, Relative File, and Indexed File) ... 113
4.3.1.4 ASSIGN Clause (Sort-merge and Report Writer) ... 115
4.3.1.5 ASSIGN Clause (Presentation File) ... 115
4.3.1.6 DESTINATION Clause (Presentation File) ... 116

4.3.1.7 END KEY Clause (Presentation File) 116
4.3.1.8 FILE STATUS Clause (Sequential File, Relative File, Indexed File, Presentation File(*), and Report Writer) 117
4.3.1.9 FORMAT Clause (Sequential File, Presentation File) 118
4.3.1.10 GROUP Clause (Sequential File, Presentation File) 118
4.3.1.11 LOCK MODE Clause (Sequential File, Relative File, and Indexed File) 119
4.3.1.12 MESSAGE CLASS Clause (Presentation File) 120
4.3.1.13 MESSAGE CODE Clause (Presentation File) 120
4.3.1.14 MESSAGE MODE Clause (Presentation File) 121
4.3.1.15 MESSAGE OWNER Clause (Presentation File) 122
4.3.1.16 MESSAGE SEQUENCE Clause (Presentation File) 122
4.3.1.17 ORGANIZATION Clause (Sequential File) 123
4.3.1.18 ORGANIZATION Clause (Relative File) 123
4.3.1.19 ORGANIZATION Clause (Indexed File) 123
4.3.1.20 ORGANIZATION Clause (Presentation File(*) and Report Writer) 123
4.3.1.21 PADDING CHARACTER Clause (Sequential File) 124
4.3.1.22 PROCESSING CONTROL Clause (Presentation File) 124
4.3.1.23 PROCESSING MODE Clause (Presentation File) 124
4.3.1.24 PROCESSING TIME Clause (Presentation File) 125
4.3.1.25 RECORD DELIMITER Clause (Sequential File) 126
4.3.1.26 RECORD KEY Clause (Indexed File) 126
4.3.1.27 RESERVE Clause (Sequential File, Relative File, Indexed File, and Report Writer) 127
4.3.1.28 SELECT Clause (Sequential File, Relative File, Indexed File, and Report Writer) 127
4.3.1.29 SELECT Clause (Sort-merge and Presentation File(*)) 127
4.3.1.30 SELECTED FUNCTION Clause (Presentation File) 128
4.3.1.31 SESSION CONTROL Clause (Presentation File) 128
4.3.1.32 SYMBOLIC DESTINATION Clause (Presentation File) 129
4.3.1.33 UNIT CONTROL Clause (Presentation File) 130
4.3.2 I-O-Control Paragraph (I-O-CONTROL) 130
4.3.2.1 APPLY MULTICONVERSATION-MODE Clause (Presentation File) 131
4.3.2.2 APPLY SAVED-AREA Clause (Presentation File) 131
4.3.2.3 MULTIPLE FILE TAPE Clause (Sequential File and Report Writer) 132
4.3.2.4 RERUN Clause (Sequential File, Relative File and Indexed File) 132
4.3.2.5 SAME Clause (Sequential, Relative, Indexed, Sort-merge, Presentation File(*), and Report Writer Files) 132

Chapter 5 Data Division 135
5.1 Composition of the Data Division 135
5.2 File Description Entry 139
5.2.1 BLOCK CONTAINS Clause (Sequential File, Relative File, Indexed File, and Report Writer Module) 140
5.2.2 CODE-SET Clause (Sequential File and Report Writer Module) 141
5.2.3 CONTROL RECORDS Clause (Sequential File) 141
5.2.4 DATA RECORDS Clause (Sequential File, Relative File, and Indexed File) 142
5.2.5 ENCODING Clause (Sequential File, Relative File, Indexed File, and Presentation File) 142
5.2.6 EXTERNAL Clause (Sequential File, Relative File, Indexed File, Presentation File(*), and Report Writer Module) 143
5.2.7 GLOBAL Clause (Sequential File, Relative File, Indexed File, Presentation File(*), and Report Writer Module) 143
5.2.8 LABEL RECORDS Clause (Sequential File, Relative File, Indexed File, and Report Writer File) 143
5.2.9 LINAGE Clause (Sequential File) 144
5.2.10 RECORD Clause (Sequential File, Relative File, and Indexed File) 146
5.2.11 RECORD Clause (Presentation File) 149
5.2.12 RECORD Clause (Report Writer Module) 149
5.2.13 REPORT Clause (Report Writer Module) 150
5.2.14 VALUE OF Clause (Sequential File, Relative File, Indexed File, and Report Writer Module) 150
5.3 Sort-Merge File Description Entry 150
5.4 Data Description Entry 151
5.4.1 BASED ON Clause 154
5.4.2 BLANK WHEN ZERO Clause 155
5.4.3 CHARACTER TYPE Clause 155
5.4.4 ENCODING Clause 159

5.4.5 EXTERNAL Clause ... 161
5.4.6 GLOBAL Clause ... 162
5.4.7 JUSTIFIED Clause ... 162
5.4.8 OCCURS Clause ... 163
5.4.9 PICTURE Clause ... 165
5.4.10 PRINTING POSITION Clause ... 176
5.4.11 REDEFINES Clause ... 177
5.4.12 RENAMES Clause ... 178
5.4.13 SIGN Clause ... 179
5.4.14 SYNCHRONIZED Clause ... 180
5.4.15 TYPE Clause ... 182
5.4.16 TYPEDEF Clause ... 183
5.4.17 USAGE Clause ... 184
5.4.18 VALUE Clause ... 191
5.5 Screen Data Description Entry ... 193
5.5.1 AUTO Clause ... 196
5.5.2 BACKGROUND-COLOR Clause ... 196
5.5.3 BELL Clause ... 197
5.5.4 BLANK LINE Clause ... 197
5.5.5 BLANK SCREEN Clause ... 197
5.5.6 BLANK WHEN ZERO Clause ... 198
5.5.7 BLINK Clause ... 198
5.5.8 COLUMN NUMBER Clause ... 199
5.5.9 ERASE Clause ... 199
5.5.10 FOREGROUND-COLOR Clause ... 200
5.5.11 FULL Clause ... 201
5.5.12 HIGHLIGHT Clause ... 201
5.5.13 JUSTIFIED Clause ... 202
5.5.14 LINE NUMBER Clause ... 202
5.5.15 LOWLIGHT Clause ... 203
5.5.16 PICTURE Clause ... 203
5.5.17 REQUIRED Clause ... 205
5.5.18 REVERSE-VIDEO Clause ... 205
5.5.19 SECURE Clause ... 205
5.5.20 SIGN Clause ... 206
5.5.21 UNDERLINE Clause ... 206
5.5.22 USAGE Clause ... 206
5.5.23 VALUE Clause ... 207
5.6 Report Description Entry ... 207
5.6.1 CODE Clause ... 209
5.6.2 CONTROL Clause ... 209
5.6.3 PAGE Clause ... 210
5.7 Report Group Description Entry ... 212
5.7.1 COLUMN NUMBER Clause ... 215
5.7.2 GROUP INDICATE Clause ... 215
5.7.3 LINE NUMBER Clause ... 216
5.7.4 NEXT GROUP Clause ... 217
5.7.5 SIGN Clause ... 217
5.7.6 SOURCE Clause ... 218
5.7.7 SUM Clause ... 218
5.7.8 TYPE Clause ... 220
5.7.9 USAGE Clause ... 224
5.7.10 VALUE Clause ... 224
5.8 Report Group Presentation Rules ... 225
5.8.1 How to Use the Presentation Table Rules ... 225
5.8.2 Report Heading Group Presentation Rules ... 226
5.8.3 Page Heading Group Presentation Rules ... 228

5.8.4 Report Body Group Presentation Rules....229
5.8.5 Page Footing Group Presentation Rules....231
5.8.6 Report Footing Group Presentation Rule....232

Chapter 6 Procedure Division....234
6.1 Composition of the Procedure Division....234
6.2 Procedure Division Header....238
6.3 Common Statement Rules....240
6.3.1 Arithmetic Expressions....240
6.3.2 Boolean Expressions....242
6.3.3 Conditional Expressions....244
6.3.3.1 Relation Condition....244
6.3.3.2 Class Condition....245
6.3.3.3 Condition-name....247
6.3.3.4 Switch-status Condition....248
6.3.3.5 Sign Condition....248
6.3.3.6 Complex Condition....248
6.3.3.7 Abbreviating a Combined Relation Condition....254
6.3.4 Comparison Rules....255
6.3.5 Rules for Moving (Transcribing) Data....260
6.3.6 Arithmetic Statements....263
6.3.7 More Than One Arithmetic Result....264
6.3.8 ROUNDED Phrase....264
6.3.9 ON SIZE ERROR Phrase....265
6.3.10 CORRESPONDING Phrase....266
6.3.11 Overlapping of Operands....266
6.3.12 INVALID KEY Phrase....266
6.3.13 AT END Phrase....267
6.3.14 Incompatible Data....269
6.4 Statements....269
6.4.1 ACCEPT Statement (Nucleus)....269
6.4.2 ACCEPT Statement (Screen Handling)....271
6.4.3 ACCEPT Statement (Command Line Arguments and Environmental Variables)....272
6.4.4 ADD Statement (Nucleus)....273
6.4.5 ALTER Statement (Nucleus)....274
6.4.6 CALL Statement (Inter-program Communication)....275
6.4.7 CANCEL Statement (Inter-program Communication)....280
6.4.8 CLOSE Statement (Sequential, Relative, Indexed, Presentation File(*), and Report Writer Module)....281
6.4.9 COMPUTE Statement (Nucleus)....284
6.4.10 CONTINUE Statement (Nucleus)....285
6.4.11 DELETE Statement (Relative and Indexed Files)....285
6.4.12 DISPLAY Statement (Nucleus)....286
6.4.13 DISPLAY Statement (Screen Handling)....288
6.4.14 DISPLAY Statement (Command Line Arguments and Environmental Variables)....289
6.4.15 DIVIDE Statement (Nucleus)....290
6.4.16 ENTRY Statement (Inter-program Communication)....293
6.4.17 EVALUATE Statement (Nucleus)....294
6.4.18 EXIT Statement (Nucleus)....298
6.4.19 EXIT PERFORM Statement (Nucleus)....298
6.4.20 EXIT PROGRAM Statement (Inter-program Communication)....299
6.4.21 GENERATE Statement (Report writer)....299
6.4.22 GO TO Statement (Nucleus)....301
6.4.23 IF Statement (Nucleus)....301
6.4.24 INITIALIZE Statement (Nucleus)....303
6.4.25 INITIATE Statement (Report writer)....305
6.4.26 INSPECT Statement....306
6.4.27 MERGE Statement (Sort-merge)....314

6.4.28 MOVE Statement (Nucleus) 319
6.4.29 MULTIPLY Statement (Nucleus) 321
6.4.30 OPEN Statement (Sequential File, Relative File, Indexed File) 322
6.4.31 OPEN Statement (Presentation File) 326
6.4.32 OPEN Statement (Report Writer) 327
6.4.33 PERFORM Statement (Nucleus) 328
6.4.34 READ Statement (Sequential File, Relative File, Indexed File) 336
6.4.35 READ Statement (Presentation File) 343
6.4.36 RELEASE Statement (Sort-merge) 345
6.4.37 RETURN Statement (Sort-merge) 345
6.4.38 REWRITE Statement (Sequential File, Relative File, Indexed File) 346
6.4.39 SEARCH Statement (Nucleus) 349
6.4.40 SET Statement (Nucleus) 354
6.4.41 SORT Statement (Sort-merge) 356
6.4.42 START Statement (Relative File) 361
6.4.43 START Statement (Indexed File) 362
6.4.44 STOP Statement (Nucleus) 367
6.4.45 STRING Statement (Nucleus) 367
6.4.46 SUBTRACT Statement (Nucleus) 373
6.4.47 SUPPRESS Statement (Report Writer) 375
6.4.48 TERMINATE Statement (Report Writer) 375
6.4.49 UNLOCK Statement (Sequential File, Relative File, Indexed File) 376
6.4.50 UNSTRING Statement (Nucleus) 376
6.4.51 USE Statement (Sequential, Relative, Indexed, Presentation File, and Report Writer Module) 385
6.4.52 USE BEFORE REPORTING Statement (Report Writer) 387
6.4.53 USE FOR DEAD-LOCK Statement 388
6.4.54 WRITE Statement (Sequential File) 389
6.4.55 WRITE Statement (Relative and Indexed Files) 394
6.4.56 WRITE Statement (Presentation File) 397
6.5 General Rules for Functions 398
6.5.1 Function Call Format 398
6.5.2 Types of Arguments 398
6.5.3 Rules Applied for Specifying a Table as an Argument 398
6.5.4 Function Types 399
6.6 Functions 400
6.6.1 ACOS Function 400
6.6.2 ADDR Function 400
6.6.3 ANNUITY Function 401
6.6.4 ASIN Function 401
6.6.5 ATAN Function 402
6.6.6 CAST-ALPHANUMERIC Function 402
6.6.7 CHAR Function 403
6.6.8 COS Function 403
6.6.9 CURRENT-DATE Function 404
6.6.10 DATE-OF-INTEGER Function 404
6.6.11 DAY-OF-INTEGER Function 405
6.6.12 DISPLAY-OF Function 405
6.6.13 FACTORIAL Function 406
6.6.14 INTEGER Function 406
6.6.15 INTEGER-OF-DATE Function 407
6.6.16 INTEGER-OF-DAY Function 407
6.6.17 INTEGER-PART Function 408
6.6.18 LENG Function 408
6.6.19 LENGTH Function 409
6.6.20 LOG Function 410
6.6.21 LOG10 Function 410
6.6.22 LOWER-CASE Function 411

6.6.23 MAX Function....411
6.6.24 MEAN Function....412
6.6.25 MEDIAN Function....412
6.6.26 MIDRANGE Function....413
6.6.27 MIN Function....413
6.6.28 MOD Function....414
6.6.29 NATIONAL Function....414
6.6.30 NATIONAL-OF Function....415
6.6.31 NUMVAL Function....415
6.6.32 NUMVAL-C Function....416
6.6.33 ORD Function....417
6.6.34 ORD-MAX Function....417
6.6.35 ORD-MIN Function....418
6.6.36 PRESENT-VALUE Function....419
6.6.37 RANDOM Function....419
6.6.38 RANGE Function....420
6.6.39 REM Function....420
6.6.40 REVERSE Function....421
6.6.41 SIN Function....421
6.6.42 SQRT Function....422
6.6.43 STANDARD-DEVIATION Function....422
6.6.44 STORED-CHAR-LENGTH Function....422
6.6.45 SUM Function....423
6.6.46 TAN Function....423
6.6.47 UCS2-OF Function....424
6.6.48 UPPER-CASE Function....424
6.6.49 UTF8-OF Function....425
6.6.50 VARIANCE Function....425
6.6.51 WHEN-COMPLIED Function....426

Chapter 7 Source Text Manipulation....428
7.1 COPY Statement....428
7.2 REPLACE Statement....433

Chapter 8 Database (SQL)....436
8.1 Format of Embedded SQL....436
8.1.1 Overall Rules....436
8.1.2 Continuation of a Line....436
8.1.3 COBOL Comment Line and In-line Comment....437
8.2 Data Division....437
8.2.1 Embedded SQL Declare Section....437
8.2.2 Host Variable Definitions....437
8.2.3 Referencing Host Variables....440
8.2.4 SQLSTATE/SQLCODE....440
8.2.5 SQLMSG....440
8.2.6 SQLERRD....440
8.3 Procedure Division....441
8.3.1 Character....442
8.3.2 Literals....442
8.3.2.1 Numeric Literal....442
8.3.2.2 Character String Literal....443
8.3.2.3 National Character String Literal....444
8.3.3 Token....444
8.3.4 Names....445
8.3.4.1 Table Name....445
8.3.4.2 Cursor Name....445
8.3.4.3 Column Name....445
8.3.4.4 Correlation Name....445

8.3.4.5 SQL Statement Identifier....445
8.3.4.6 Stored Procedure Names....445
8.3.5 Value Specification and Target Specification....446
8.3.6 Column Specification....447
8.3.7 Set Function Specification....448
8.3.8 Value Expression....448
8.3.9 Predicate....449
8.3.9.1 Comparison Predicate....449
8.3.9.2 BETWEEN Predicate....450
8.3.9.3 IN Predicate....450
8.3.9.4 LIKE Predicate....451
8.3.9.5 NULL Predicate....451
8.3.9.6 Quantified Predicate....451
8.3.9.7 EXISTS Predicate....452
8.3.10 Search Condition....452
8.3.11 Table Expression....452
8.3.11.1 FROM Clause....453
8.3.11.2 WHERE Clause....453
8.3.11.3 GROUP BY Clause....453
8.3.11.4 HAVING Clause....453
8.3.12 Query Specification....454
8.3.13 Query Expression....454
8.3.14 Subquery....454
8.3.15 FOR Clause....455
8.4 Embedded Exception Declaration....455
8.5 Data Manipulation without Using Cursor....456
8.5.1 SELECT Statement....456
8.5.2 DELETE Statement (Searched)....457
8.5.3 INSERT Statement....457
8.5.4 UPDATE Statement (Searched)....458
8.6 Data Manipulation Using the Cursor....459
8.6.1 Declare Cursor....459
8.6.2 OPEN Statement....460
8.6.3 CLOSE Statement....461
8.6.4 FETCH Statement....461
8.6.5 DELETE Statement (Positioned)....462
8.6.6 UPDATE Statement (Positioned)....462
8.7 Dynamic SQL....463
8.7.1 INTO Clause / USING Clause....463
8.7.2 PREPARE Statement....463
8.7.3 EXECUTE Statement....464
8.7.4 EXECUTE IMMEDIATE Statement....465
8.7.5 Dynamic SELECT Statement....466
8.7.6 Dynamic Declare Cursor....466
8.7.7 Dynamic OPEN Statement....466
8.7.8 Dynamic CLOSE Statement....467
8.7.9 Dynamic FETCH Statement....467
8.7.10 Dynamic DELETE Statement (Positioned)....468
8.7.11 Dynamic UPDATE Statement (Positioned)....468
8.8 Transaction Management....469
8.8.1 COMMIT Statement....469
8.8.2 ROLLBACK Statement....469
8.9 Connection Management....469
8.9.1 CONNECT Statement....469
8.9.2 SET CONNECTION Statement....470
8.9.3 DISCONNECT Statement....471
8.10 Stored Procedure....471

8.10.1 The CALL Statement....471

Chapter 9 Communication Database....472
9.1 Basic Elements of Embedded DCSQL....473
9.1.1 Available Characters....473
9.1.2 Quotation Marks, Key Words, and Separators....473
9.1.3 Communication Database Names....473
9.1.4 Service Names....474
9.1.5 Table Names....474
9.1.6 Host Variable Names....474
9.1.7 Literals....474
9.1.7.1 Character String Literal....474
9.1.7.2 National Character String Literal....475
9.1.7.3 Exact Numeric Literal....475
9.1.7.4 Approximate Literal....475
9.2 Reference Format of Embedded DCSQL....476
9.2.1 Overall Rules for Description....476
9.2.2 Continuation of Line....476
9.2.3 COBOL Comment Line and In-line Comment....476
9.2.4 Comments in Embedded DCSQL....476
9.3 Embedded DCSQL....476
9.3.1 Data Division....476
9.3.1.1 Embedded DCSQL Declare Section....476
9.3.1.2 Host Variable Definitions....477
9.3.1.3 Referencing Host Variables....478
9.3.1.4 DCSQLSTATE....478
9.3.1.5 DCSQLMSG....479
9.3.2 Procedure Division....479

Chapter 10 Micro Focus Native Functions....480
10.1 Named Literal....480
10.1.1 User-defined Words....480
10.1.2 Scope of Name....480
10.1.3 Data Description Entry....480
10.2 Hexadecimal Numeric Literal....481
10.3 Screen Functions....481
10.3.1 Environment Division....482
10.3.1.1 Special-names Paragraph....482
10.3.1.1.1 CONSOLE IS CRT Clause....482
10.3.2 Data Division....482
10.3.2.1 Screen Data Description Entries....482
10.3.2.1.1 COLUMN NUMBER Clause....485
10.3.2.1.2 CONTROL Clause....485
10.3.2.1.3 GRID Clause....486
10.3.2.1.4 LEFTLINE Clause....487
10.3.2.1.5 LINE NUMBER Clause....487
10.3.2.1.6 OCCURS Clause....487
10.3.2.1.7 OVERLINE Clause....487
10.3.2.1.8 PROMPT Clause....488
10.3.2.1.9 SIZE Clause....488
10.3.2.1.10 ZERO-FILL Clause....489
10.3.3 Procedure Division....489
10.3.3.1 ACCEPT Statement (Screen Operation)....489
10.3.3.2 DISPLAY Statement (Screen Operation)....492
10.4 National Functions....495
10.4.1 Procedure Division....495
10.4.1.1 Class Condition....495
10.5 Input-Output Statement....496

10.5.1 Procedure Division ... 496
10.5.1.1 READ Statement (Relative File and Indexed File) ... 496
10.5.1.2 START Statement (Relative File) ... 500
10.5.1.3 START Statement (Indexed File) ... 500
10.6 Inter-program Communication Function ... 502
10.6.1 PROCEDURE DIVISION ... 502
10.6.1.1 CALL Statement (Inter-program Communication) ... 503

Chapter 11 Object-Oriented Programming Functions ... 505
11.1 Overview ... 505
11.2 Terminology ... 505
11.3 General Rules ... 508
11.3.1 Basic Elements of the Language ... 508
11.3.1.1 COBOL Words ... 508
11.3.1.1.1 User-Defined Words ... 508
11.3.1.1.2 Reserved Words ... 509
11.3.1.2 Figurative Constants ... 509
11.3.2 Concept of Data Description Entry ... 509
11.3.2.1 Concept of Class ... 509
11.3.3 Uniqueness of Reference ... 510
11.3.3.1 Identifier ... 510
11.3.3.2 In-Line Method Invocation ... 512
11.3.3.3 Object-Modifier ... 513
11.3.3.4 Predefined Object Identifier ... 514
11.3.3.4.1 EXCEPTION-OBJECT ... 514
11.3.3.4.2 NULL ... 515
11.3.3.4.3 SELF and SUPER ... 515
11.3.3.5 Object Property ... 515
11.3.3.6 Class-Name ... 516
11.3.4 Reference Format ... 516
11.3.4.1 Continuation of Lines ... 517
11.3.4.2 Scope of Names ... 517
11.4 Compilation Group Structure ... 517
11.4.1 Organization ... 517
11.4.1.1 COBOL Compilation Group ... 518
11.4.2 External Repository ... 520
11.4.3 Program Organization and Communication ... 521
11.4.3.1 Objects and Classes ... 521
11.4.3.2 Object References ... 521
11.4.3.3 Methods ... 521
11.4.3.4 File Connector ... 521
11.4.3.5 External Name and Internal Name ... 521
11.4.3.6 Global Names ... 522
11.4.3.7 Method Invocation ... 522
11.4.3.8 Method Results ... 522
11.4.3.9 Special Registers ... 522
11.4.3.10 Interfaces and Conformance ... 522
11.4.3.11 Polymorphism ... 522
11.4.3.12 Class Inheritance ... 523
11.4.4 Life Cycle for Objects ... 523
11.4.5 Life Cycle for Factory Objects ... 523
11.5 Identification Division and End Marker ... 523
11.5.1 Organization of the IDENTIFICATION DIVISION ... 523
11.5.1.1 CLASS-ID Paragraph ... 524
11.5.1.2 FACTORY Paragraph ... 524
11.5.1.3 OBJECT Paragraph ... 524
11.5.1.4 METHOD-ID Paragraph ... 524

11.5.1.5 AUTHOR Paragraph....526
11.5.1.6 INSTALLATION Paragraph....526
11.5.1.7 DATE-WRITTEN Paragraph....526
11.5.1.8 DATE-COMPILED Paragraph....526
11.5.1.9 SECURITY Paragraph....526
11.5.2 END Marker....527
11.5.2.1 The END CLASS Header....527
11.5.2.2 The END FACTORY Header....527
11.5.2.3 The END OBJECT Header....528
11.5.2.4 The END METHOD Header....528
11.6 The ENVIRONMENT DIVISION....528
11.6.1 The CONFIGURATION SECTION....528
11.6.1.1 The REPOSITORY Paragraph....529
11.6.2 The INPUT-OUTPUT SECTION....530
11.6.2.1 The I-O-CONTROL Paragraph....531
11.7 The DATA DIVISION....531
11.7.1 The FILE SECTION....531
11.7.2 The LINKAGE SECTION....531
11.7.3 File Description Entries....531
11.7.3.1 The LINAGE Clause....532
11.7.4 Data Description Entries....532
11.7.4.1 The ANY LENGTH clause....532
11.7.4.2 The CHARACTER TYPE clause....533
11.7.4.3 The EXTERNAL Clause....533
11.7.4.4 The GLOBAL Clause....534
11.7.4.5 The OCCURS Clause....534
11.7.4.6 The PRINTING POSITION Clause....534
11.7.4.7 The PROPERTY Clause....534
11.7.4.8 The REDEFINES Clause....536
11.7.4.9 The RENAMES clause....537
11.7.4.10 The TYPE clause....537
11.7.4.11 The TYPEDEF clause....537
11.7.4.12 The USAGE clause....537
11.7.4.12.1 The USAGE IS INDEX clause....537
11.7.4.12.2 The USAGE IS OBJECT REFERENCE clause....537
11.7.4.13 The VALUE Clause....539
11.8 The PROCEDURE DIVISION....539
11.8.1 Composition of the PROCEDURE DIVISION....539
11.8.2 PROCEDURE DIVISION Header....540
11.8.3 Common Rules Applicable to Statements....542
11.8.3.1 Invocation Operator....542
11.8.3.2 Relation Condition between Object Identifiers....542
11.8.4 Execution....542
11.8.4.1 Status of Methods and Objects....542
11.8.4.1.1 Status of Methods....542
11.8.4.1.2 State of Data Associated Only with an Object....542
11.8.4.2 Explicit and Implicit Control Flow....543
11.8.4.3 Manipulation of Conditions....543
11.8.4.3.1 Exception Conditions....543
11.8.4.3.2 Exception Objects....543
11.8.5 Conformance....543
11.8.5.1 Interfaces of a Class....544
11.8.5.2 Interfaces of an Object....544
11.8.5.3 Conformance between Interfaces....544
11.8.5.4 Conformance for Assignment....544
11.8.5.5 Conformance of Parameters....547
11.8.5.5.1 Conformance of Group Items....547

11.8.5.5.2 Conformance of Elementary Items to be passed BY CONTENT ... 547
11.8.5.5.3 Conformance of Elementary Items to be passed by BY REFERENCE ... 548
11.8.5.5.4 Conformance when The ANY LENGTH Clause Is Specified as the Receiving Item ... 548
11.8.5.6 Conformance of Returning Items ... 548
11.8.5.6.1 Conformance of Group Items ... 548
11.8.5.6.2 Conformance of Elementary Items ... 548
11.8.6 Statements ... 550
11.8.6.1 The ALTER Statement (Object-Oriented Programming) ... 550
11.8.6.2 The CALL Statement (Object-Oriented Programming) ... 550
11.8.6.3 The ENTRY Statement (Object-Oriented Programming) ... 550
11.8.6.4 The EVALUATE Statement (Object-Oriented Programming) ... 550
11.8.6.5 The EXIT Statement (Object-Oriented Programming) ... 550
11.8.6.6 The GO TO Statement (Object-Oriented Programming) ... 551
11.8.6.7 The INITIALIZE Statement (Object-Oriented Programming) ... 551
11.8.6.8 The INVOKE Statement (Object-Oriented Programming) ... 552
11.8.6.9 The MERGE Statement (Object-Oriented Programming) ... 554
11.8.6.10 The OPEN Statement (Object-Oriented Programming) ... 554
11.8.6.11 The RAISE Statement (Object-Oriented Programming) ... 554
11.8.6.12 The SET Statement (Object-Oriented Programming) ... 555
11.8.6.13 The SORT Statement (Object-Oriented Programming) ... 555
11.8.6.14 The STOP Statement (Object-Oriented Programming) ... 555
11.8.6.15 The USE Statement (Object-Oriented Programming) ... 555
11.8.6.16 The USE FOR DEAD-LOCK Statement (Object-Oriented Programming) ... 556
11.8.6.17 The WRITE Statement (Object-Oriented Programming) ... 556
11.9 Database (SQL) ... 556
11.9.1 Reference Format of Embedded SQL ... 556
11.9.2 DATA DIVISION ... 557
11.9.2.1 Embedded SQL Declarative Section ... 557
11.9.3 Embedded Exception Declarative ... 557
11.9.4 Cursor System Data Operation Statement ... 557
11.9.4.1 Cursor Declarative ... 557
11.9.5 Dynamic SQL ... 557
11.9.5.1 EXECUTE Statement ... 557
11.9.5.2 Dynamic Cursor Declarative ... 557
11.10 COBOL System Class ... 558
11.10.1 FJBASE Class ... 558
11.10.1.1 The CREATE Method ... 558
11.10.1.2 The NEW Method ... 558
11.10.1.3 The GETCLASS Method ... 559
11.10.1.4 The INIT Method ... 559
11.10.1.5 The _FINALIZE Method ... 559
11.10.2 NULL Classes and NULL Objects ... 559

Chapter 12 Microsoft .NET Support ... 560
12.1 Overview ... 560
12.2 Terminology ... 560
12.3 Additions to Basic Elements of COBOL ... 564
12.3.1 COBOL Words ... 564
12.3.1.1 User-Defined Words ... 564
12.3.1.2 Reserved Words ... 565
12.3.2 Unsupported Words ... 566
12.3.3 Literal ... 566
12.3.3.1 Numeric Literals ... 566
12.3.4 Uniqueness of Reference ... 567
12.3.4.1 Identifier ... 567
12.3.4.2 In-line Method Invocation ... 569
12.3.4.3 Object Modifiers ... 570

12.3.4.4 Predefined Object Identifiers....571
12.3.4.4.1 EXCEPTION-OBJECT....571
12.3.4.4.2 NULL....571
12.3.4.4.3 SELF and SUPER....572
12.3.4.5 Object Property....572
12.3.4.6 TYPE OF Operator....573
12.3.4.7 Field References....573
12.3.5 External and Internal Names....574
12.3.5.1 External Names....575
12.3.5.1.1 Name modification with namespaces or nesting....575
12.3.5.1.2 Modification for Array Types....575
12.3.5.1.3 Modification for Type Parameters....576
12.4 Compilation Group Structure....576
12.4.1 Organization....576
12.4.1.1 COBOL Compilation Group....577
12.4.2 Generics....581
12.5 Identification Division....581
12.5.1 PROGRAM-ID Paragraph....582
12.5.2 CLASS-ID Paragraph....583
12.5.3 STATIC Paragraph....584
12.5.4 OBJECT Paragraph....585
12.5.5 METHOD-ID Paragraph....585
12.5.6 DELEGATE-ID Paragraph....588
12.5.7 INTERFACE-ID Paragraph....589
12.5.8 ENUM-ID Paragraph....589
12.5.9 CUSTOM-ATTRIBUTE Phrase....590
12.5.10 END Markers....591
12.6 Environment Division....592
12.6.1 CONFIGURATION SECTION....592
12.6.1.1 SPECIAL-NAMES Paragraph....592
12.6.1.2 CUSTOM-ATTRIBUTE Clause....593
12.6.1.3 REPOSITORY Paragraph....594
12.6.2 INPUT-OUTPUT Section....598
12.6.3 I-O-CONTROL Paragraph....598
12.7 Data Division....599
12.7.1 FILE SECTION....599
12.7.2 SCREEN SECTION....599
12.7.3 REPORT SECTION....599
12.7.4 LINKAGE SECTION....599
12.7.5 Data Description Entry....599
12.7.5.1 CUSTOM-ATTRIBUTE Phrase....600
12.7.5.2 CHARACTER TYPE Clause....601
12.7.5.3 EXTERNAL Clause....601
12.7.5.4 GLOBAL Clause....601
12.7.5.5 OCCURS Clause....601
12.7.5.6 PRINTING POSITION Clause....602
12.7.5.7 PROPERTY Clause....602
12.7.5.8 REDEFINES Clause....604
12.7.5.9 USAGE Clause....604
12.7.5.10 VALUE Clause....607
12.7.5.11 Accessibility Specifier Phrase (PUBLIC/PRIVATE/PROTECTED)....608
12.8 Procedure Division....608
12.8.1 PROCEDURE DIVISION Header....609
12.8.2 Common Statement Rules....611
12.8.2.1 Conditional Expressions....611
12.8.2.1.1 Class Condition....611
12.8.2.1.2 Type Condition....613

12.8.2.2 Comparisons ... 613
12.8.2.3 Rules for Moving (Transcribing) Data ... 616
12.8.2.3.1 Parameter Conformance ... 616
12.8.2.4 Exception Conditions ... 633
12.8.2.4.1 Exception Object ... 633
12.8.2.5 ON SIZE ERROR Phrase ... 633
12.8.3 Statements ... 633
12.8.3.1 Conditional statements and imperative statements ... 633
12.8.3.2 ALTER ... 633
12.8.3.3 CALL ... 633
12.8.3.4 DISPLAY ... 636
12.8.3.5 ENTRY ... 636
12.8.3.6 EXIT ... 636
12.8.3.7 GO TO ... 637
12.8.3.8 INVOKE ... 638
12.8.3.9 PERFORM ... 641
12.8.3.10 RAISE ... 642
12.8.3.11 RESUME ... 643
12.8.3.12 SET ... 643
12.8.3.13 STOP ... 652
12.8.3.14 TRY ... 652
12.8.3.15 USE ... 653
12.9 Functions ... 654
12.9.1 ACP-OF Function ... 654
12.9.2 DISPLAY-OF Function ... 654
12.9.3 ENUM-OR Function ... 655
12.9.4 ENUM-AND Function ... 655
12.9.5 ENUM-NOT Function ... 656
12.9.6 NATIONAL-OF Function ... 656
12.9.7 UNICODE-OF Function ... 657
12.10 Intermediate Results ... 657
12.10.1 Intermediate Results of Arithmetic Operations ... 657

Appendix A List of Reserved Words ... 658

Appendix B System Quantitative Restrictions ... 679
B.1 Reference Format ... 679
B.2 Data Division ... 679
B.3 Procedure Division ... 679
B.4 Sequential File ... 680
B.5 Relative File ... 680
B.6 Indexed File ... 680
B.7 Inter-program Communication ... 680
B.8 Sort and Merge ... 681
B.9 Source Statement Manipulation ... 681
B.10 Presentation File ... 681
B.11 Object-Oriented Programming ... 681

Appendix C Code Tables ... 682
C.1 Internal Codes for EBCDIC Characters ... 682
C.2 Internal Codes for ASCII Characters ... 683
C.3 Internal Codes for JIS 8-Bit Code Characters ... 683

Appendix D Intermediate Results ... 685
D.1 Attribute and Precision of the Intermediate Result ... 685
D.2 Intermediate Results of Arithmetic Operations ... 685
D.2.1 Precision of the Intermediate Results of Arithmetic Operations Having the Fixed Point Attribute ... 686
D.2.2 Intermediate Result in Arithmetic Operations Having the Floating-point Attribute ... 690

D.3 Intermediate Result of the Exponent....690
D.3.1 Intermediate Result of the Exponent Having Fixed-point Attribute....691
D.3.2 Intermediate Result of the Exponent Having Floating-point Attribute....691
D.4 Attributes and Accuracy of Function Values....691

Appendix E Functional Differences....694
E.1 Reference Format for Free Coding Format....694
E.2 Sequential Files....694
E.3 Data Item Definition....694
E.4 Fixed-point literals....695
E.5 Presentation Files....695
E.6 Inter-Program Communication....695
E.7 Statements....696
E.8 Functions....697
E.9 Database....697
E.10 Communication Database....698
E.11 Micro Focus Native Function....698
E.12 Object-oriented Programming....698
E.13 .NET Programming Function....698
E.14 Quantitative System Limits....698

Appendix F Control Record Formats....700
F.1 I Control Record....700
F.2 S Control Records....703

Appendix G Defining a Data Item Using a Type....705
G.1 Type....705
G.1.1 Type of an elementary item....705
G.1.2 Type of a group item....706
G.2 Strong Type....707
G.3 Type Referencing Other Types....709
G.4 Effectively Using the Data Definition using a Type....711

Index....712

Chapter 1 General Rules

This chapter explains the concepts of the COBOL language and covers the general rules.

1.1 Characters and Character Sets

The most basic and indivisible unit of the COBOL language is the character. The sets used to form COBOL character-strings and separators are called "COBOL character sets". There are four types of COBOL character sets:

- Alphabetic character set
- Numeric character set
- Special character set
- National character set

F

Alphabetic Character Set

The alphabetic characters are listed below. Each alphabetic character uses 1 byte of storage.

- Uppercase letters (26 letters from A to Z)
- Lowercase letters (26 letters from a to z)
- Space

Except in nonnumeric literals, lowercase letters are equivalent to their corresponding uppercase letters.

Numeric Character Set

There are ten numeric characters: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Each numeric character uses 1 byte of storage.

Special Character Set

The table below lists the special characters. Each special character uses 1 byte of storage.

Character	Meaning
+	Plus sign
-	Minus sign or hyphen
*	Asterisk
/	Slash
=	Equal sign
$	Dollar sign
,	Comma
;	Semicolon
.	Period or decimal point
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater-than sign
<	Less-than sign
:	Colon

Character	Meaning
&	Ampersand
_	Underscore

F Underscore is specific to [Win32], [Winx64], [.NET], [Solaris], [Linux], [LinuxIPF] and [Linux64].

F

National Characters

National characters are used with the Japanese character set. Each character in the character set is expressed by a two-byte internal code.

Unless otherwise specified, alphabetic or numeric characters do not contain national characters.

Computer Character Set

Character sets used on a computer are called "computer character sets". COBOL character sets incorporate a portion of a computer's character set.

Characters from COBOL character sets and from other computer character sets can be used in the following items:

- Nonnumeric literal
- F - National nonnumeric literal
- Comment entry
- Comment line
- F - In-line comment

1.2 Basic Overview of the Language

A COBOL program is made up of

- separators,
- COBOL words,
- literals,
- character strings in PICTURE clauses,
- comment entries,
- comment lines,
- F - and in-line comments.

The COBOL character sets are used for writing all program elements except for literal values, comment entries, comment lines, and in-line comments.

These elements are written using characters from a computer character set.

A COBOL program can also contain a line consisting of all spaces (blank line).

1.2.1 Separator

A separator is a string of one or more punctuation characters that is used to delimit character strings. Use any of the following characters as a separator:

- A sequence of one or more spaces. This is called a "separator space".
- A sequence of one comma followed by one or more spaces. This is called a "separator comma".
- A sequence of one semicolon followed by one or more spaces. This is called a "separator semicolon".
- A sequence of one period followed by one or more spaces. This is called a "separator period".

- A left parenthesis
- A right parenthesis
- A quotation mark indicating the start of a literal value (")
- A quotation mark indicating the start of a literal value (X", NC", N", NX" or B")
- A quotation mark indicating the end of the literal value
- The == pseudo-text delimiter
- A colon
- The -> pointer qualification symbol
- The & concatenation operator

Separator Space

The separator space delimits a COBOL word, literal, or character string in a PICTURE clause.

Separator Comma and Separator Semicolon

The separator comma and separator semicolon facilitate program reading. Either one can be used wherever a separator space can be used.

A separator space can also be included immediately before or after these separators.

Separator Period

A separator period is used to indicate the end of the declaration a division, section, paragraph or sentence. A period can only be written wherever a period (".") is used in a syntax diagram.

A separator space can be included immediately before or after a period.

Left and Right Parentheses

Use left and right parentheses to enclose :

- subscript,
- reference modifier,
- arithmetic expression,
- Boolean expression,
- conditional expression, or
- argument list.

Parentheses are always used in pairs.

A separator space can be included immediately before or after the left or right parenthesis.

Quotation Mark

Quotation marks are used to enclose :

- nonnumeric literal,
- hexadecimal nonnumeric literal,
- National nonnumeric literal,
- National hexadecimal nonnumeric literal, or
- a character string of a Boolean literal value.

Quotation marks are always used in pairs, one mark indicating the start of a literal value and the other mark indicating the end of the literal value. Rules for describing quotation marks are:

1. Enclose a nonnumeric literal with a pair of quotation marks.
2. Enclose a hexadecimal nonnumeric literal with X" and a quotation mark.
3. Enclose a National nonnumeric literal with NC" and a quotation mark or N" and a quotation mark.
4. Enclose a National hexadecimal nonnumeric literal with NX" and a quotation mark.
5. Enclose a Boolean literal with B" and a quotation mark.

In all cases a separator space or a left parenthesis must immediately precede the quotation mark indicating the start of a literal value. Also, a separator space, separator comma, separator semicolon, separator period, or right parenthesis must immediately follow the quotation mark indicating the end of the literal value.

Pseudo-text Delimiter

Pseudo-text delimiters enclose pseudo-text. They are used in pairs. A separator space must immediately precede the pseudo-text delimiter indicating the start of pseudo-text. A separator space, separator comma, separator semicolon, or separator period must immediately follow the pseudo-text delimiter indicating the end of pseudo-text.

Colon

A colon is used to write a reference modifier. A separator space can be included immediately before or after the colon.

Pointer Qualification Symbol

A pointer qualification symbol is used to write a pointer qualifier. A separator space can be included immediately before or after the colon.

Concatenation Operator

A concatenation operator is used in concatenation expressions. A separator space must be written immediately before and after the concatenation operator.

Character-strings Not Regarded as Separators

The rules for separators do not apply to character-strings written in the locations listed below. A character-string written in any of the following locations is not regarded as a separator even if the character-string has the same format as a separator:

- Numeric literal
- A portion enclosed by a separator indicating the start of a literal value and a separator indicating the end of a literal value in :
  - nonnumeric literal,
  - hexadecimal nonnumeric literal,
  - National nonnumeric literal,
  - or Boolean literal
- A character string in a PICTURE clause
- A comment entry
- A comment line
- An in-line comment

1.2.2 COBOL Word

A COBOL word is a single character string consisting of COBOL character set characters. There are four types of COBOL words:

- User-defined words
- System-name

- Function-name
- Reserved words

Each COBOL word has its own restrictions in addition to the following general restrictions:

1. A COBOL word must not exceed 30 characters in length.
2. A COBOL word must be made up of :
   - alphabetic characters (A to Z and a to z),
   - numeric characters (0 to 9),
   - hyphens (-),
   - or underscores (_).

   It must not contain any spaces. Lowercase letters are regarded as being equivalent to their corresponding uppercase letters. A user-defined word may consist of national characters.
3. A hyphen must not be used as the first or last character of a COBOL word.

   An underscore must not be used as the first or last character of a COBOL word.

Note

Names that begin with an underscore are COBOL reserved words.

1.2.2.1 User-defined Words

A user-defined word is any of the following 20 words as named by the user:

- Positioning unit name
- Print mode name
- Symbolic-constant
- Symbolic-character
- Text-name
- Index-name
- Class-name
- Condition-name
- Section-name
- Paragraph-name
- Data-name
- Library-name
- File-name
- Alphabet-name
- Program-name
- Report-name
- Mnemonic-name
- Record-name
- Level-number

- Type-name

The names of all user-defined words must be unique, except level-number. However, condition-names, data-names, record-names, and index-names may have the same names provided qualification allows for referential uniqueness. The same level number may be written multiple times in one program.

1.2.2.1.1 Rules for Describing User-defined Words

A user-defined word is made up of :

- alphabetic characters,
- numeric characters,
- hyphens,
- underscores,
- or national characters.

User defined words except level-numbers, library-names, program-names, and text-names may be expressed using national characters.

Program-names may be expressed using national characters in [Win32], [Winx64], [.NET], [Solaris], [Linux], [LinuxIPF] and [Linux64]. User-defined words composed of national characters are called "National user-defined words."

The following rules must be satisfied when creating a user-defined word from

- alphabetic characters,
- numeric characters,
- hyphens,
- and underscores.

1. The word must not exceed 30 characters in length.
2. The word must be made up of alphabetic characters (A to Z and a to z), numeric characters (0 to 9), hyphens (-), or underscores (_).

   It must not contain any spaces. Lowercase letters are regarded as equivalent to their corresponding uppercase letters.
3. A hyphen and underscore must not be used as the first or last character of the word.
4. A character string representing a user-defined word must not be identical to a reserved word, but may be identical to a function-name or system-name.
5. All user-defined words except paragraph-names, section-names, and level-numbers must include at least one alphabetic character.
6. A user-defined word must not contain two or more consecutive underscores.

The following rules must be satisfied when creating a user-defined word from national characters:

1. The word must not exceed 30 characters in length.
2. The word must be made up of national characters and contain no spaces. Lowercase letters in JIS non-kanji characters are not regarded as being equivalent to their corresponding uppercase letters in JIS non-kanji characters.
3. A JIS non-kanji hyphen or JIS non-kanji minus sign must not be used as the first or last character of the word.
4. The word must include at least one JIS level-1 kanji character, JIS level-2 kanji character, JIS non-kanji hiragana character, JIS non-kanji katakana character, or JIS non-kanji macron.
5. Level-numbers, library-names, program-names, and text-names cannot be written in national characters. However, in [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET], program-names can contain national characters.

1.2.2.1.2 Application of User-defined Words

The application for user-defined words and the locations where they can be written are explained below.

F

Positioning unit name

A positioning unit name is the name given to the value indicating the column to be printed. It is defined in the in the POSITIONING UNIT clause of the SPECIAL-NAMES paragraph.

F

Print mode name

A print mode name is the name given to the presentation format to be printed, and specifies such things as character size, pitch, typeface, rotation, and configuration. It is defined in the PRINTING MODE clause of the SPECIAL-NAMES paragraph.

F

Symbolic-constant

A symbolic-constant is the name given to a literal for referencing the literal by name. It is defined in the SYMBOLIC CONSTANT clause of the SPECIAL-NAMES paragraph.

Symbolic-character

A symbolic-character is the name given to a character used to define words that can be used like figurative constants. It is defined in the SYMBOLIC CHARACTER clause of the SPECIAL-NAMES paragraph.

Text-name

A text name is the name given to library text for referencing a COBOL library. It can be specified in the COPY statement.

Index-name

An index-name is the name given to an index to a table. It is defined in the DATA DIVISION paragraph.

Class-name

A class name is the name given to a set of characters and can be used to test whether or not a data item contains any of the specified characters. It is defined in the CLASS clause of the SPECIAL-NAMES paragraph.

Condition-name

There are two types of condition names:

1. The name given to an assumed value of a data item
2. The name given to the ON or OFF status of an external switch

The first type of condition-name is defined in a condition-name data description entry. A data item associated with a condition-name is called a "conditional variable". This type of condition-name can be used in a condition-name condition or in a SET statement. A condition-name condition is an abbreviated format of a relation condition. It indicates whether the value of the conditional variable is equal to the condition-name value. A condition-name may be used in a SET statement for setting the conditional variable value.

The second type of condition-name is defined in the SPECIAL-NAMES paragraph and represents the status of an external switch. The switch-status condition indicates the status of an external switch, that is, whether it is set ("on") or not ("off"). If a mnemonic-name is specified for the condition-name, the SET statement may be used to set the status of an external switch.

Section-name

A section-name is the name given to a PROCEDURE DIVISION section.

Paragraph-name

A paragraph-name is the name given to a PROCEDURE DIVISION paragraph.

Data-name

A data-name is the name given to a data item. It is defined in a data description. When "data-name-n" is indicated in a syntax diagram, do not use

- reference modification,
- subscripting,
- qualification,
- or an explicit pointer

except where specifically permitted. When "identifier-n" is indicated in a syntax diagram, reference modification, subscripting, qualification, or an explicit pointer may be required when referencing a particular data-name.

Library-name

A library-name is the name given to a COBOL library for referencing a COBOL library.

File-name

A file-name is the name given to a file connector. It is defined in the ENVIRONMENT DIVISION FILE-CONTROL entry.

Alphabet-name

An alphabet-name is the name given to a specified character code set and / or collating sequence. It is defined in the ALPHABET clause of SPECIAL-NAMES paragraph.

Program-name

A program-name is the name given to a COBOL source program. It is defined in the PROGRAM-NAME clause of the IDENTIFICATION DIVISION.

Report-name

A report-name is the name given to a report. It is defined in the REPORT SECTION of the DATA DIVISION.

Mnemonic-name

A mnemonic-name associates a user-defined name with an implementer-name. It is defined in the SPECIAL-NAMES paragraph.

Record-name

A record-name is the name given to a record. It is defined in a data division record description entry.

Level-number

A level number is the number given to indicate the data item, report group item, or screen item hierarchy. A level number between 01 and 49 indicates the item location in the hierarchical structure of a record. Use level-number 66, 77, or 88 when defining a data description entry having special characteristics. The high-order zero in level-numbers 01 to 09 may be omitted.

F

Type-name

Type-name is the name given to data types. A data type is defined in the data description entry as a type declaration. Type-name is a function specific to [Win32], [Winx64], [.NET], [Solaris], [Linux], [LinuxIPF] and [Linux64].

1.2.2.2 System Name

A system-name links the program with the operating system. A system-name can be either a user-specified word or a specific word.

The following system-names are used:

- Function-name
- Computer-name
- Language-name
- File-identifier-name

Function Name

A function-name identifies resources defined outside the program such as a logical device on the system, layout of printed pages, and external switches. When using a resource defined externally, function-name must be associated with a mnemonic-name in the SPECIAL-NAMES paragraph.

A function-name is different for each resource. See the topic "Special-names Paragraph" in the section "Composition of the Environment Division" of Chapter 4, the "Environment Division" for the rules pertaining to the description of a function-name.

Computer-name

A computer-name identifies the computer compiling or running the program.

The user specifies a computer-name by writing a character string conforming to COBOL words rules.

Language-name

A language-name identifies a particular programming language.

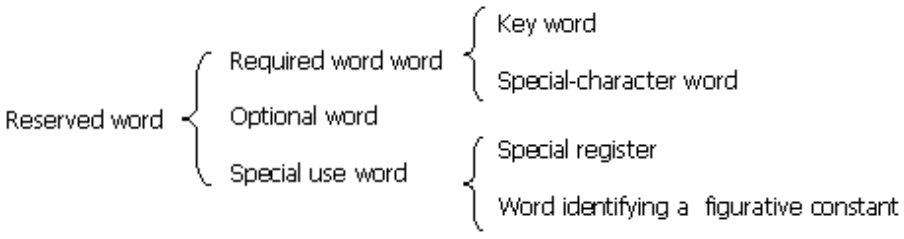
File-identifier-name

A file-identifier-name is a name for identifying a file on a storage medium. The file-identifier-name must be associated with a file-name in the ENVIRONMENT DIVISION FILE-CONTROL entry before the file may be used.

See the topics beginning with the title "ASSIGN Clause (Sequential file, Relative file, and Indexed File) " and ending with the title "ASSIGN clause (Presentation File) " in the "Input-Output Section" section of Chapter 4, the "Environment Division" for rules on describing a file-identifier-name.

1.2.2.3 Reserved Words

A reserved word is a COBOL word that is one of a specified list of words that may be used in COBOL source programs, but must not appear in programs as user-defined words or System-names. There are several types of reserved words as shown below.



See Appendix A, "List of Reserved Words", for the reserved words list.

Required word

A required word is a word that must be included. There are two types of required words:

- Key word
- Special-character word

A key word is written in uppercase letters and underlined in all syntax diagrams.

A special-character word is one of the twelve characters shown below. It is not underlined in a syntax diagram in order to avoid confusion with other symbols.

+, - , * , / , ** , < , > , = , >= , <= , &, ->

Optional word

An optional word may be written or omitted as desired. It appears in uppercase letters in syntax diagrams and is not underlined.

Special use word

There are two types of special use words:

- Special register
- Figurative constant

Special register

A special register is a storage area automatically generated by the compiler. It is used to store information produced when using specific COBOL features.

There are several types of special registers:

- A special register for use with sequential files
  - LINAGE-COUNTER

    Note that this is not available in [.NET].

F
- Special registers for use with the Inter-program communication module
  - PROGRAM-STATUS
  - RETURN-CODE (synonym for PROGRAM-STATUS)
- Special registers for use with the Sort-Merge module
  - SORT-STATUS
  - SORT-CORE-SIZE
- Special registers for use with the Presentation File module
  - EDIT-MODE
  - EDIT-OPTION
  - EDIT-OPTION2
  - EDIT-OPTION3
  - EDIT-COLOR
  - EDIT-STATUS
  - EDIT-CURSOR
- Special registers for use with the Report Writer module
  - LINE-COUNTER
  - PAGE-COUNTER

F
- Special registers for use with the format 3 MOVE statement.
  - CONV-STATUS
  - CONV-SIZE

Special Registers are data items generated by the COBOL system and are referred to through the use of their associated names. Special register names can be used wherever "data-name-n" or "identifier-n" is specified in a syntax diagram.

F

EDIT-OPTION2 and EDIT-OPTION3 are specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

CONV-STATUS and CONV-SIZE are specific to [Win32], [Winx64] and [Linux64].

figurative constant

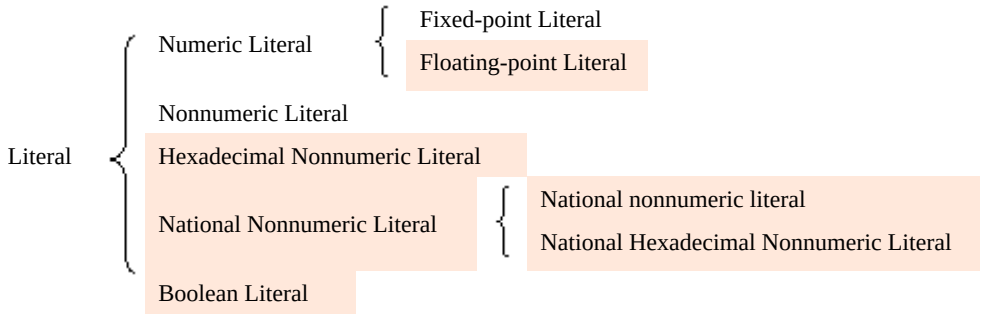
A figurative constant is used to refer to specific constant values, and is expressed using reserved words. For example, ALL and SPACE in the figurative constant ALL SPACE are reserved words.

1.2.2.4 FUNCTION-NAME

A function-name is a word indicating the name of a function. It can be written in a function-identifier clause. The function type determines the location where the function-identifier can be written. See the "General Rules for Functions" section of Chapter 6, the "Procedure Division" for more information on the location where a function-identifier can be used.

1.2.3 Literal

A literal is data having a constant value. There are several literal types as shown below.



1.2.3.1 Numeric Literal

A numeric literal is a character-string whose characters represent a numeric value. There are two types of numeric literals:

- Fixed-point literal

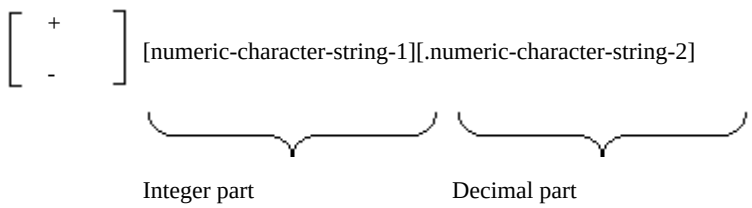
F

- Floating-point literal

Fixed-point Literal

A fixed-point literal is expressed by a combination of signs, numeric characters, and a decimal point.

Format



1. A fixed-point literal consists of a sign, integer part, decimal point, and decimal part.
2. The sign "+" represents the plus sign, "-" represents the minus sign, and "." represents the decimal point.
3. Numeric-character-string-1 and numeric-character-string-2 must contain numeric characters between 0 and 9 only.

4. The total number of digits in the integer part and decimal part must be as shown below:

F

   - 1 to 18 for [DS], [HP] and [Win16].
   - 1 to 19 for [Win32], [Solaris], [Linux], [LinuxIPF] and [.NET].
   - In [Winx64] and [Linux64], 1 to 19 for 18-digit compatibility operation mode. 1 to 31 for 31-digit extension operation mode

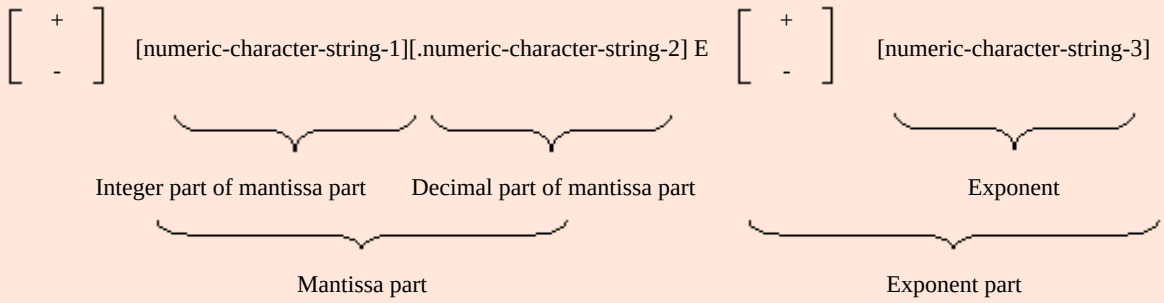
5. The fixed-point literal value is algebraic.
6. The fixed-point literal category is numeric.
7. A fixed-point literal can be written in the following locations:
   - A fixed-point literal having a decimal point may be specified wherever "literal-n" is indicated in a syntax diagram, and where a numeric literal is permitted.
   - A fixed-point literal not containing a decimal point can be specified wherever "literal-n" is indicated in syntax diagram and where a numeric literal is permitted, or it can be specified wherever "integer-n" is indicated in syntax diagram. Only unsigned integers of 1 or more may be specified where "integer-n" is indicated in a syntax diagram, unless otherwise permitted in the syntax rules.

F

Floating-point Literal

A floating-point literal is expressed in the format "mantissa*(10**exponent) ".

Format



1. A floating-point literal consists of a mantissa part, "E", and an exponent part. The mantissa part consists of a sign, an integer part, a decimal point and a decimal part. The exponent part consists of a sign and an exponent.
2. The sign "+" represents the plus sign, "-" represents the minus sign, and "." represents the decimal point.
3. Numeric-character-string-1 to numeric-character-string-3 must contain numeric characters between 0 and 9 only.
4. The total number of digits in the integer part and decimal part of the mantissa part must be between 1 and 16.
5. The number of digits in the exponent must be 1 or 2.
6. The value of the floating-point literal must be an algebraic value represented by the following formula:

```
Mantissa part * (10 ** exponent part)
```

7. The floating-point literal category is numeric.
8. A floating-point literal can be written wherever "literal-n" is indicated in a syntax diagram, and where a numeric literal is permitted.

1.2.3.2 Nonnumeric Literal

A nonnumeric literal is a character-string whose value represents a character constant.

Format

"{ character-1 } ... "

1. The string in character-1 must be delimited at both ends with quotation marks.
2. Any character belonging to a computer character set may be used as character-1.
3. Use double quotation marks to represent a single quotation mark in a literal.
4. The length of the nonnumeric literal must be between 1 and 160 bytes.
5. The value of a nonnumeric literal is the string of characters itself. The separator quotation marks enclosing the nonnumeric literal are not part of the value of the nonnumeric literal.
6. The category of a nonnumeric literal is alphanumeric.
7. A nonnumeric literal can be used wherever "literal-n" is indicated in a syntax diagram, and where a nonnumeric literal is permitted.

F

1.2.3.3 Hexadecimal Nonnumeric Literal

A hexadecimal nonnumeric literal is a character-string whose value represents a hexadecimal value. A hexadecimal nonnumeric literal represents the literal value as a hexadecimal character.

Format

X"{ hexadecimal-character-1 } ... "

1. The characters in hexadecimal-character-1 must be delimited by the separator X" at the left and by double quotation marks at the right.
2. Hexadecimal-character-1 must consist of characters between 0 and 9 or A to F. Two consecutive characters in hexadecimal-character-1 represent a single character code or part of a character code from the computer character set.
3. The number of characters in hexadecimal-character-1 must be between 2 and 320.
4. The value of a hexadecimal nonnumeric literal is the value (a character belonging to the computer character set) indicated in the string of hexadecimal-character-1. A hexadecimal nonnumeric literal is regarded as equivalent to a nonnumeric literal.
5. The hexadecimal nonnumeric literal category is alphanumeric.
6. A hexadecimal nonnumeric literal can be specified wherever "literal-n is indicated in a syntax diagram, and where a nonnumeric literal is permitted. In this reference manual the term "nonnumeric literal" also includes a hexadecimal nonnumeric literal.
7. The table below shows the internal bit configuration of hexadecimal characters.

Hexadecimal Character	Internal Bit Configuration
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011

Hexadecimal Character	Internal Bit Configuration
C	1100
D	1101
E	1110
F	1111

F

1.2.3.4 National Nonnumeric Literal

A national nonnumeric literal is a character-string whose value is represented by national characters. There are two types of national nonnumeric literal:

- National nonnumeric literal
- National hexadecimal nonnumeric literal

National nonnumeric literal

A National nonnumeric literal represents a literal value using the national character set.

Format 1

NC"{ national-character-1 } ... "

Format 2

N"{ national-character-1 } ... "

1. Format 1 and format 2 are equivalent.
2. The characters in national-character-1 must be delimited by the separator NC" or the separator N" at the left and by double quotation marks at the right.
3. Any national character belonging to a computer character set can be used as national-character-1.
   In [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET], however, when the operation mode is Unicode (ISO/IEC 10646-1), any character belonging to a computer character set can be used.
4. The national-character-1 count must be between 1 and 80.
5. The National nonnumeric literal value is the national characters specified in the national-character-1 list.
6. The National nonnumeric literal category is national language.
7. A National nonnumeric literal can be used wherever "literal-n" is indicated in a syntax diagram and where a National nonnumeric literal is permitted.

National Hexadecimal Nonnumeric Literal

A national hexadecimal nonnumeric literal represents a literal value as a hexadecimal character.

Format

NX"{ hexadecimal-character-1 } ... "

1. The characters in hexadecimal-character-1 must be delimited by the separator NX" at the left and by double quotation marks at the right.
2. Hexadecimal-character-1 must consist of characters between 0 and 9 or A to F. Two consecutive characters in hexadecimal-character-1 represent a single national character code from a computer character set.
3. The hexadecimal-character-1 count must be between 4 and 320.

4. The national hexadecimal nonnumeric literal value is the value (a national character belonging to a computer character set) indicated in the hexadecimal-character-1 list. A national hexadecimal nonnumeric literal is regarded as equivalent to a National nonnumeric literal.
5. The category of a national hexadecimal nonnumeric literal is national language.
6. A national hexadecimal nonnumeric literal can be specified wherever "literal-n" is indicated in a syntax diagram and where a national nonnumeric literal is permitted.

F

1.2.3.5 Boolean Literal

A Boolean literal is a literal value represented by a Boolean character.

Format

B"{ Boolean-character-1 } ... "

1. The characters in Boolean-character-1 must be delimited by the separator B" at the left and by double quotation marks at the right.
2. The Boolean-character-1 must consist of characters "0" and "1" only.
3. The Boolean-character-1 count must be between 1 and 160.
4. The Boolean literal category is Boolean.
5. The Boolean literal value is the value written in the Boolean-character-1 list.
6. A Boolean literal can be specified wherever "literal-n" is indicated in a syntax diagram and where a Boolean literal is permitted.

1.2.4 Figurative Constant

A figurative constant represents a literal having a specific value. There are seven types of figurative constants:

- ZERO
- SPACE
- HIGH-VALUE
- LOW-VALUE
- QUOTE
- ALL literal
- Symbolic-character

The following table shows the format and value for each figurative constant.

Name	Format	Value
ZERO	[ALL] ZERO [ALL] ZEROS [ALL] ZEROES	Has one of the following values depending on context: - Numeric value zero - Repeats character "0" from a computer character set. - Repeats Boolean character "0".
SPACE	[ALL] SPACE [ALL] SPACES	Has one of the following values depending on context: - An alphabetic or national character space from a computer character set - Repeats an alphabetic national character space from a computer character set.

Name	Format	Value
HIGH-VALUE	[ALL] HIGH-VALUE [ALL] HIGH-VALUES	Has one of the following values depending on context: - The highest character code in the collating sequence of a program - Repeats the highest character code in the collating sequence of a program.
LOW-VALUE	[ALL] LOW-VALUE [ALL] LOW-VALUES	Has one of the following values depending on context: - The smallest character code in the collating sequence of a program - Repeats the smallest character code in the collating sequence of a program.
QUOTE	[ALL] QUOTE [ALL] QUOTES	Has one of the following values depending on context: - A quotation mark. - Repeats a quotation mark
ALL literal	ALL literal	Repeats the literal written after ALL.
Symbolic-character	[ALL] Symbolic-character	Has one of the following values depending on context: - A symbolic-character - Repeats a symbolic-character.

1. ALL must be immediately followed by a separator space.
2. Singular and plural forms such as SPACE and SPACES are synonymous.
3. Quotation marks enclosing the value in
   - nonnumeric literal,
   - hexadecimal nonnumeric literal,
   - national nonnumeric literal or
   - Boolean literal

   cannot be expressed by the figurative constant QUOTE. For example, "ABD" cannot be written as QUOTE ABD QUOTE.
4. A literal written with the ALL literal clause must be
   - nonnumeric literal,
   - hexadecimal nonnumeric literal,
   - national nonnumeric literal,
   - Boolean literal, or
   - an associated symbolic literal.

   A figurative constant cannot be specified in an ALL literal clause.
5. A symbolic-character is defined in the SPECIAL-NAMES paragraph SYMBOLIC CHARACTERS clause.
6. A figurative constant may be specified wherever "literal-n" is indicated in a syntax diagram, unless otherwise indicated.
7. Any transcription or relation between an ALL literal in which ALL is immediately followed by a literal of two or more characters and a numeric data item or numeric edited data item becomes an obsolete element. Avoid using this element when generating a new program.
8. When a figurative constant that is composed of a character-string covering several positions is specified in a VALUE clause, or when a transcription or relation is positioned between such a figurative constant and a data item, the figurative constant is regarded as being the same length as the data item. That is, the character-string specified in the

figurative constant is repeated until it is as long as the data item. If it becomes longer than the data item, the character-string is truncated from the right so it is equal in length to the data item. The figurative constant value is transcribed before the JUSTIFIED clause in the data item is processed.

F

1.2.5 Concatenation Expression

A concatenation expression is a method of expressing the value of a single literal by combining two or more literals with a concatenation operator.

Refer to "Concatenation Expressions" in the "COBOL Syntax Samples", for an example of the usage.

Format

[ALL] { literal-1 } { & literal-2 } ...

1. The ampersand (&) is called a concatenation operator. The concatenation operator must be immediately preceded and followed by a separator space.
2. The combination of literal-1 and literal-2 must be one of the following:
    a. A combination consisting of at least one nonnumeric literal, hexadecimal nonnumeric literal, figurative constant SPACE, figurative constant HIGH-VALUE, figurative constant LOW-VALUE, figurative constant QUOTE, figurative constant ZERO, or a symbolic-character. The literal category expressed in this concatenation expression is alphanumeric.
    b. A combination of national nonnumeric literal or a combination of a national nonnumeric literal and a figurative constant SPACE. The literal category expressed in this concatenation expression is national language.
3. ALL must not be affixed to the figurative constant when writing a figurative constant in literal-1 or literal-2.
4. Either of the following values may be used as the figurative constant value when using a figurative constant in literal-1 or literal-2:
    a. When using at least one nonnumeric or hexadecimal nonnumeric literal in a concatenation expression, or when using a figurative constant in a concatenation expression, the figurative constant value can be expressed as a single alphanumeric character specified in the figurative constant.
    b. When using at least one national nonnumeric literal in a concatenation expression, the figurative constant value can be expressed as a single national character specified in the figurative constant.
5. The concatenation expression value is the value obtained when combining the literal values in the concatenation expression.
6. The maximum length of the literal resulting from the concatenation expression is 160 characters.
7. When using a figurative constant in a concatenation expression, use the keyword ALL. A concatenation expression containing ALL is treated in the same manner as an ALL literal.
8. A concatenation expression may be specified wherever "literal-n" is indicated in a syntax diagram and where a concatenation expression is permitted.
9. A comment entry or blank line can appear between the concatenation operator and a literal.

F

1.2.6 Literals for Special Applications

There are several types of literals for special applications:

- Program-name literal
- File-identifier literal
- Text-name literal

Program-name Literal

A program-name literal is a literal whose value represents a program-name.

The program-name literal value must conform to the rules for program-names as determined by the system. For more information on the rules for program-name literals, refer to the "NetCOBOL User's Guide".

File-identifier Literal

A file-identifier literal is a literal whose value represents the file-identifier of a sequential, relative, or indexed file.

The format of the file-identifier literal is shown below.

```
"name"
```

Specify the file name and the storage medium in "name".

A file-identifier literal must be written in accordance with the rules for nonnumeric literals. The value and length of a file-identifier literal must conform to the rules determined by the system. For more information on the rules for file-identifier literals, refer to the "NetCOBOL User's Guide".

A file-identifier literal can be used wherever "file-identifier literal-n" is indicated in a syntax diagram.

Text-name Literal

A text-name literal is a literal whose value represents a library name.

A text-name literal must be expressed in accordance with the rules for text-name literals. The value and length of the text-name literal must conform to the rules determined by the system. For more information on the rules for text-name literals, refer to the "NetCOBOL User's Guide".

A text-name literal can be used wherever "text-name-literal-n" is indicated in a syntax diagram.

1.2.7 Picture Character-string

A PICTURE character-string is a character-string written in a PICTURE clause. A PICTURE character-string consists of alphabetic characters, numeric characters, and special characters belonging to a COBOL character set. The topic titled "PICTURE clause" in section "Data Description Entry" of Chapter 5, the "Data Division" lists the rules for describing a PICTURE character-string.

1.2.8 Comment Entry

A comment entry is an IDENTIFICATION DIVISION entry. It consists of any characters belonging to the computer character set.

A comment entry is an obsolete element. Avoid using this element when creating new programs.

1.3 Concept of Data Description

This section explains the data description concept.

1.3.1 Concept of Levels

The most comprehensive collection of data consisting of at least one piece of data is called a "record".

A record is configured according to the concept of levels. This concept arose from the need to subdivide a record for referencing data. The record can be subdivided for referencing data.

The most basic portion of a record, that is, the portion which cannot be subdivided further, is called an " elementary item". Several elementary items may be grouped so they can be referenced collectively, this is known as a "group item". Furthermore, several groups can be grouped. An elementary item can therefore be a subordinate to several groups.

1.3.2 Concept of Class

Elementary items other than index data items, pointer data items, and internal floating-point data items all belong to a category and class. The data class for a group item is regarded as alphanumeric character, regardless of the elementary item categories belonging to it.

The relationship between category and data class is shown in the following table.

Level of Item	Type	Category	Class
Elementary item	Alphabetic data item	Alphabetic character	Alphabetic character
	Numeric data item	Numeric	Numeric
	Numeric edited data item	Numeric edited	Alphanumeric character
	Alphanumeric data item	Alphanumeric character	
	Alphanumeric edited data item	Alphanumeric edited	
	National data item	National	National
	National edited data item	National edit	
	Boolean data item	Boolean	Boolean
	External floating-point data item	External floating	Alphanumeric character
Group item	-	Alphabet character Numeric Numeric edited Alphanumeric character Alphanumeric edited	Alphanumeric character
		National National edit Boolean External floating-point item	

F

1.3.3 Concept of Types

- A type is a template that contains all the characteristics of data items and their subordinates. A type is declared using the TYPEDEF clause. The type name defines the essential characteristics of the type. The characteristics described are the lengths, relative positions, PICTURE, USAGE, SIGN, SYNCHRONIZED, JUSTIFIED, and BLANK WHEN ZERO clauses of elementary items defined by the type declaration.
- A type is referenced by specifying the TYPE clause in the data description entry. The typed item defined by this specification has all the characteristics of the referenced type.
- Items can be weakly or strongly typed. Elementary items may be weakly-typed only, group items may be either weakly or strongly typed.

Usage of types is specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET]. However, strong types cannot be used in [.NET].

1.3.3.1 Weakly-Typed Item

Weakly-typed items have the characteristics of the referenced type. These characteristics used can be the same as non-typed items except that they cannot be overwritten by the high-order group items of the typed item. Thus, a weakly-typed declaration can be regarded as a "shorthand" of a series of data description entries.

1.3.3.2 Strongly-Typed Item

A strongly typed item has the characteristics of the referenced type and its data is completely protected by the following restrictions:

1. A strongly typed item shall only be a level 1 group item or a group item subordinate to a type declaration with the STRONG phrase.
2. The data description entry of a strongly typed item shall not contain a value clause.
3. A strongly typed group and its subordinate elementary items must adhere to the following rules:
    a. These items must not be redefined either explicitly or implicitly.
    b. These items must not be renamed in whole or in part.
    c. Reference modification must not be performed on an elementary item whose category is not the alphanumeric or national.
    d. Reference modification must not be performed on group items.
4. A strongly typed item can be referenced by the following:
    - Method invocation parameter, temporary parameter, or returning item
    - Relation condition
    - DISPLAY statement
    - INITIALIZE statement
    - MOVE statement (excluding the CORRESPONDING phrase)
    - READ statement
    - RELEASE statement
    - RETURN statement
    - REWRITE statement
    - WRITE statement
    - LENGTH function
    - LENG function
5. When a strongly typed item is moved, compared, or transferred, the following rules must be observed:
    - When one item references a type with the TYPE clause, the other item must reference an equivalent type.
    - When one item is a subordinate item of a referenced type, the other item must belong to an equivalent type, and begin from the same relative position and have the same length.

    Equivalent types mean that the two type declarations satisfy the following conditions:
    a. The types must be declared with the same name.
    b. When group items subordinate to type declarations have TYPE clauses, the type names referenced with the TYPE clauses must be the same.
    c. The elementary items subordinate to type declarations must begin from the same relative position and have the same length.
    d. When an elementary item subordinate to the type declaration contains the BLANK WHEN ZERO, JUSTIFIED, PICTURE, SIGN, SYNCHRONIZED, or USAGE clause, the clause information must match between the two types.
    e. When an elementary item subordinate to the type declaration contains the ENCODING clause defined explicitly or implicitly, that clause must match between the two types. The ENCODING clause is specific to [Win32], [Winx64] and [Linux64].

1.3.4 Standard Alignment Rule

The standard alignment rule when storing data in an elementary item is determined by the receiving item category. Data from the sending item is aligned in accordance with the following rules.

1. When the receiving item is a numeric data item, the data is aligned in accordance with the following rules:

   a. When the assumed decimal point in the receiving item is explicit, the decimal point in the sending item is aligned with the decimal point position in the receiving item before the data is transcribed. After transcription, the end of the receiving item is padded with zeroes or truncated, whichever is applicable.

F In this compiler, compile option TRUNC must be specified.

   b. When the assumed decimal point in the receiving item is not explicit, the assumed decimal point is regarded as being at the right end of the receiving item and data is aligned in according with the rule described in above.

   However, if the receiving item is a BINARY-CHAR/SHORT/LONG/DOUBLE data item and the value of the integer part on the sending item is not in the value range shown in the Chapter 5 table Internal Representation of BINARY-CHAR/SHORT/LONG/DOUBLE Data Items, no result is specified for the move.

2. When the receiving item is a numeric edited data item, the sending item aligns data with the decimal point position in the receiving item and then the receiving item is edited and transcribed.
   At this time, either end of the receiving item is padded or filled with zeroes or truncated, whichever is applicable, except where editing requirements cause replacement of leading zeros.

3. When the receiving area is an alphanumeric data item, alphanumeric edited data item, alphabetic data item, National data item, or national edited data item, the sending item is aligned with the left side of the receiving item and transcribed. At this time, the right end is either padded with spaces or truncated, whichever is applicable. However, if a JUSTIFIED clause was specified in the receiving item, data is aligned in accordance with the rules for JUSTIFIED clauses.

F 4. When the receiving item is Boolean, the Boolean position of the sending item low order end is aligned with the Boolean position of the receiving item low order end before data is transcribed. At this time, the end of the receiving item is padded with Boolean character 0 or truncated, whichever is applicable.

1.3.5 Adjustment of Data Boundaries

When a SYNCHRONIZED clause has been specified in a data description entry,

- an unused character position or

F - Boolean position

may be inserted by the compiler to force the data item to be aligned to a native boundary.

The character position inserted by the compiler to align data to a native boundary is called the "slack byte", respectively.

F The Boolean position inserted by the compiler to align data to a native boundary is called the "slack bit", respectively.

1.3.5.1 Slack Byte

When a SYNCHRONIZED clause is specified for a binary data item or internal floating-point data item, the data item is allocated to a native boundary in accordance with the SYNCHRONIZED clauses rules. At this time, a slack byte may be inserted into the storage area of the item.

Rules for Insertion of a Slack Byte

The compiler allocates a relative address for each data item in the computer's storage area. The leading address in the storage area is 0 and the value obtained by adding the data item size to the relative address of the data item becomes the relative address of the next data item.

When a SYNCHRONIZED clause has been specified for a binary data item or internal floating-point data item, the data item digit positions may be padded with a slack byte. A slack byte is an implicit FILLER of a minimum length required for padding.

A slack byte can be inserted in the following positions:

1. When the data item immediately preceding the data item requiring padding is an elementary item, a FILLER having the same level-number as the level-number of the data item requiring padding is inserted immediately before the data item requiring padding.
2. When the data item immediately preceding the data item requiring padding is a group item, the program searches for the item immediately following an elementary item having the highest level-number among the series of group items subordinate to the data item requiring padding. It inserts a FILLER item having the same level-number as the level-number of the group item immediately before the group item.
3. When a group item that specifies an OCCURS clause has a subordinate data item requiring padding, no slack byte is inserted if the size of one occurrence of the group data is not a multiple of the maximum value of the byte count for the data item native boundary subordinate to the group item. A slack byte is inserted after each occurrence of the group item.

When a slack byte is inserted, the size of the group item is expanded by the size of the implicit FILLER inserted in the group item.

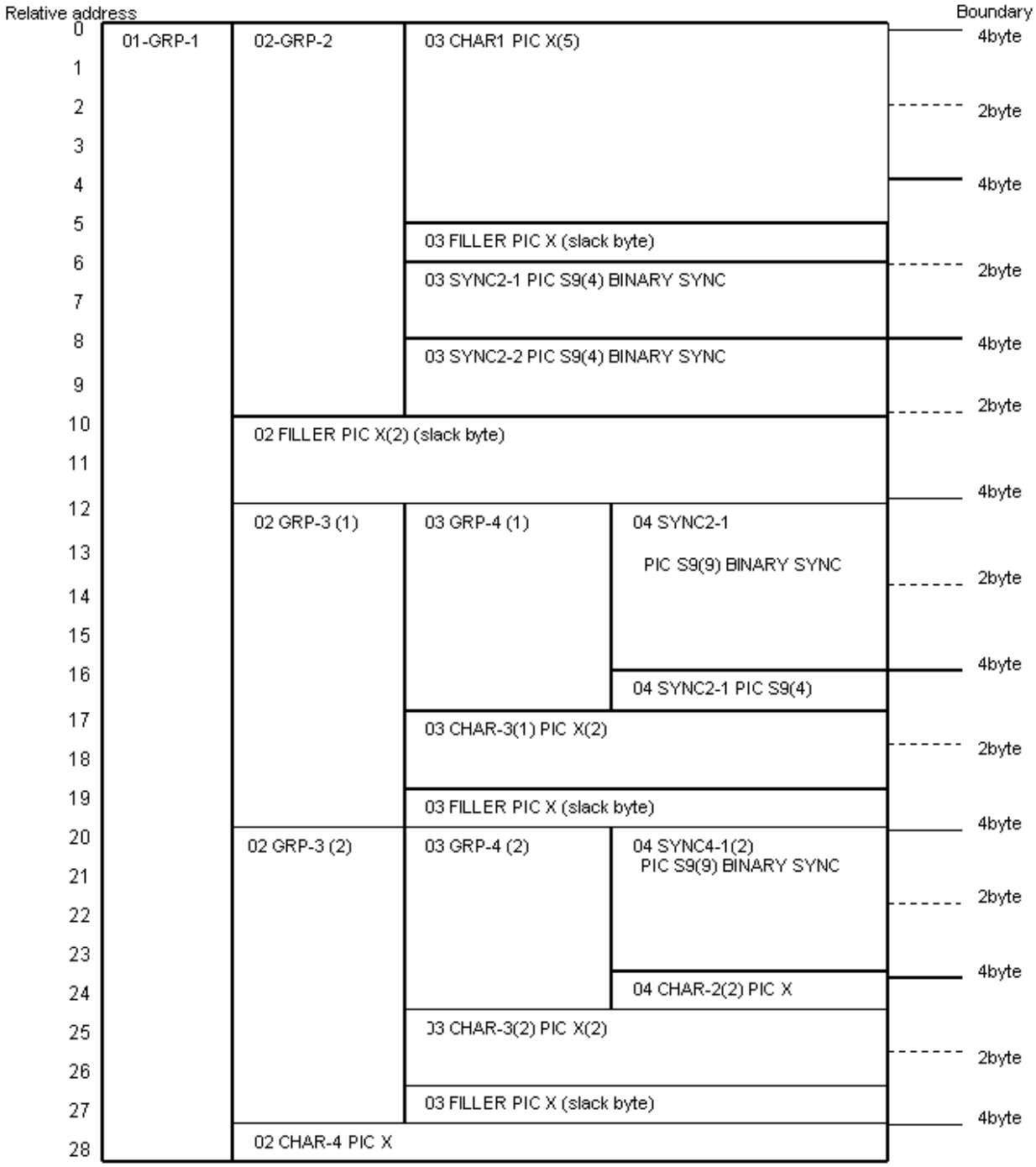
Example

Example of Insertion of a Slack Byte

Below is an example of slack byte insertion for the following record description entry.

```
01  GRP-1.
  02  GRP-2.
    03  CHAR-1   PIC X(5).
    03  SYNC2-1  PIC S9(4)  BINARY SYNC.
    03  SYNC2-2  PIC S9(4)  BINARY SYNC.
  02  GRP-3    OCCURS 2.
    03  GRP-4.
       04  SYNC4-1  PIC S9(9)  BINARY SYNC.
       04  CHAR-2   PIC X.
    03  CHAR-3     PIC X(2).
  02  CHAR-4       PIC X.
```

The diagram below shows area allocation of the above record description entry.



F

1.3.5.2 Slack Bit

When a SYNCHRONIZED clause has been specified for a Boolean data item, the computer' s storage area is divided into units of one byte and begins with a one-byte boundary allocated to the Boolean data item. A slack bit may be inserted into the storage area for the item.

Rules for Insertion of a Slack Bit

A slack bit is inserted in accordance with the following rules:

1. A slack bit is inserted in one of the following positions:

    a. For a Boolean data item in which the SYNCHRONIZED clause has been specified, the slack bit is inserted between the right end of the padded Boolean data item and the next byte boundary.

b. For data items other than a Boolean data item for which no SYNCHRONIZED clause has been specified, the slack bit is allocated to the byte boundary immediately following a Boolean data item where no SYNCHRONIZED clause record has been specified, between the right end of the internal Boolean data item and the next byte boundary.

c. When the last data item subordinate to a group item is a Boolean data item, the slack bit is inserted between the right end of the internal Boolean data item and the next byte boundary.

2. The number of slack bits is determined by the following:

   a. First, the number of bits contained in a Boolean data item is determined. For contiguous Boolean data items which have no SYNCHRONIZED clause as specified in condition (1)(b) above, the total number of Boolean bits containing Boolean data items in the sequence is determined. For contiguous Boolean data items which have no SYNCHRONIZED clause specified before the last Boolean data item in condition (1)(c) above, the total number of Boolean bits containing Boolean data items in the sequence is determined.

   b. The number of Boolean bits found at (a) is divided by 8.

   c. If the remainder from the division at (b) is zero, no slack bit is required. If the remainder is not zero, a slack bit is inserted. Assuming the remainder from the division at (b) is r, then the slack bit size is 8 - r bit(s).

3. The slack bit is a FILLER having the same level-number as the data item level-number immediately preceding the slack bit. The slack bit size is included when counting the group item size where the slack bit was inserted.

4. When a group item that specifies an OCCURS clause has a subordinate internal Boolean data item, a slack bit is inserted. Each occurrence of the table elements is regarded as a group item and the procedure in 1.c above is followed.

5. In the followings, a slack bit is automatically inserted :

   - An output file
   - WORKING-STORAGE section
   - CONSTANT section
   - LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)

   An input file and the LINKAGE SECTION are regarded as having a slack bit, so the slack bit must be accounted for when defining a record description entry.

Example

Example of Insertion of a Slack Bit (1)

Below is an example of a slack bit insertion for the following record description entry.

```
01  RECORD-A.
    02  DATA-A1.
      03  DATA-A11  PIC 1(4) BIT.
      03  DATA-A12  PIC 1(3) BIT SYNC.
      03  DATA-A13  PIC 1(5) BIT.
    02  DATA-A2  PIC 1(4) BIT.
```

DATA-A12 is allocated to a one-byte boundary, so four slack bits are inserted between DATA-A11 and DATA-A12.

A one-byte area is allocated to DATA-A12, so five slack bits are inserted between DATA-A12 and DATA-A13.

DATA-A13 is the last internal Boolean data item subordinate to DATA-A1, so three slack bits are inserted after DATA-A13.

The above record description entry is equivalent to the example shown below.

```
01  RECORD-A.
    02   DATA-A1.
      03  DATA-A11  PIC 1(4) BIT.
      03  FILLER    PIC 1(4) BIT.     *> Slack bit
      03  DATA-A12  PIC 1(3) BIT.
```

```
       03  FILLER    PIC 1(5) BIT.     *> Slack bit
       03  DATA-A13  PIC 1(5) BIT.
       03  FILLER    PIC 1(3) BIT.     *> Slack bit
     02  DATA-A2  PIC 1(4) BIT.
```

Example of Insertion of a Slack Bit (2)

Below is an example of a slack bit insertion in a group item where an OCCURS clause has been specified for the following record description entry.

```
01  RECORD-B.
    02  DATA-B1 OCCURS 10 TIMES.
      03  DATA-B11  PIC 1(5) BIT.
      03  DATA-B12  PIC 1(5) BIT.
```

DATA-B1 is allocated to begin on a one-byte boundary where DATA-B1 is repeated. Therefore, six slack bits are inserted after DATA-B12. The slack bits inserting method is the same as the repetition of DATA-B1.

When an OCCURS clause has been specified in a Boolean data item containing no SYNCHRONIZED clause, no slack bit is inserted between each table element occurrence. For example, no slack bit is inserted between each occurrence of DATA-C1 in the case shown below.

```
  02  DATA-C1 PIC 1(3) BIT OCCURS 6.
```

1.4 Uniqueness of Reference

All user-defined words assigned in a program must be unique. The method for making user-defined words unique is called "uniqueness of reference".

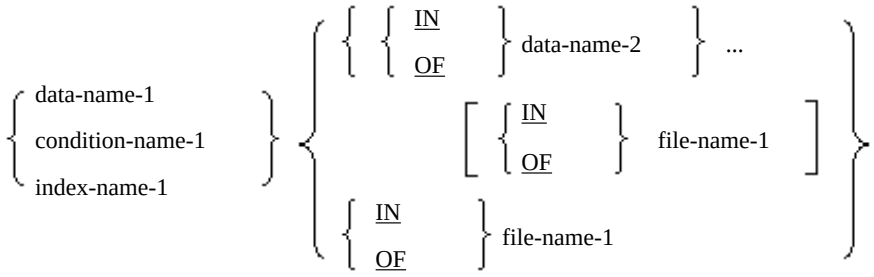
1.4.1 Qualification

If two or more user-defined words having identical spelling are in the same name scope, when referencing the user-defined words qualification must be used. This is done by adding at least one high-order name according to the name hierarchy to the reference. The topic titled "Scope of Names" in the "Inter-program Communication Module" of section of Chapter 2, "COBOL Modules" explains the scope of a name.

Refer to "Qualification" in the "COBOL Syntax Samples", for an example of the usage.

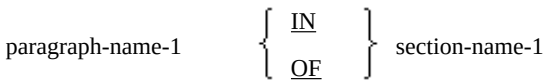
Format 1

Making a data-name, condition-name, or index-name unique:



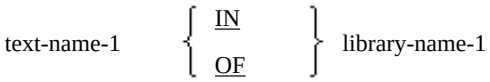
Format 2

Making a paragraph-name unique:



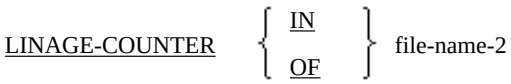
Format 3

Making a text-name unique:



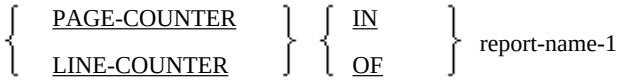
Format 4

Making the special register used for the sequential I-O module unique:



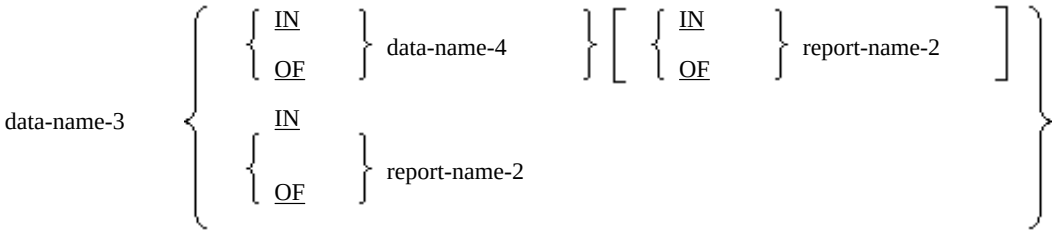
Format 5

Making the special register used with the report writer module unique:



Format 6

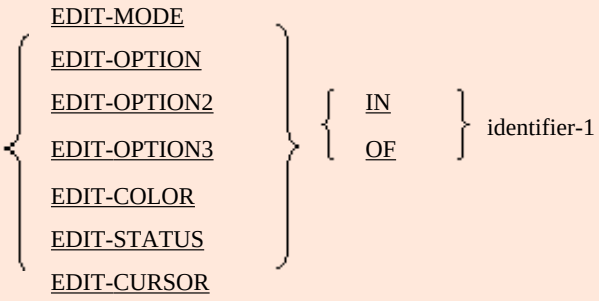
Making a data-name in a report unique:



F

Format 7

Making the special register used for the presentation file module unique:



EDIT-OPTION2 and EDIT-OPTION3 are specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

Rules Common to Format 1 to Format 7

1. IN or OF and the word following are together referred to as a "qualifier". Non-unique user-defined words must include a sequence of qualifiers before the user-defined words can be referenced.

2. Qualifiers must continue to be added to the user-defined words until it becomes unique. However, not all qualifiers must be added.
3. A user-defined word not needing qualification may be qualified.
4. IN and OF are synonymous.

Rules for Format 1

1. The qualifier must be one of the following:
    a. When qualifying data-name-1 with data-name-2, data-name-2 must be the data-name of a group item subordinate to data-name-1.
    b. When qualifying condition-name-1 with data-name-2, data-name-2 must be the data-name of a conditional variable associated with condition-name-1.
    c. When qualifying index-name-1 with data-name-2, data-name-2 must be the data description entry data-name where index-name-1 was written in the INDEXED BY phrase.
    d. When qualifying data-name-1, condition-name-1, or index-name-1 with file-name-1, file-name-1 must be the file-name of a file description entry or Sort-Merge file description entry associated with the item (data-name-1, condition-name-1, or index-name-1) to be qualified.
2. Specify qualifiers in order from low-order qualifiers to high-order qualifiers according to the hierarchy of the item going from left to right.
3. A record-name can be specified for data-name-1 or data-name-2.
4. When qualifying a condition-name, use the conditional variable hierarchy associated with the condition-name as the qualifier.

Rules for Format 2

1. When explicitly referencing a paragraph-name, the paragraph-name must not be defined twice in one SECTION.
2. When referencing a defined paragraph-name in the same section as the paragraph-name, the paragraph-name need not be qualified.
3. SECTION must not be added to section-name-1.
4. A paragraph-name or section-name cannot be referenced from another program.

Rules for Format 3

When using two or more COBOL libraries, text-name must be qualified by a library-name before being referenced.

Rules for Format 4

When the LINAGE clause is being used with two or more file description entries, LINAGE-COUNTER must be qualified with a file-name before the LINAGE-COUNTER special register can be referenced.

Rules for Format 5

1. When two or more report description entries have been defined, LINE-COUNTER must be qualified with a report-name before the LINE-COUNTER special register can be referenced.
2. When LINE-COUNTER has been referenced without first being qualified in the REPORT SECTION, LINE-COUNTER is implicitly qualified by the report-name. Before referencing LINE-COUNTER for another report in the report section, LINE-COUNTER must be explicitly qualified by the report-name.
3. When two or more report description entries have been written, PAGE-COUNTER must be qualified with a report-name before the PAGE-COUNTER special register can be referenced.
4. When PAGE-COUNTER has been referenced without first being qualified in the REPORT SECTION, PAGE-COUNTER is implicitly qualified by the report-name. Before referencing PAGE-COUNTER for another report in the report section, PAGE-COUNTER must be explicitly qualified by the report-name.

Rules for Format 6

1. The qualifier must be one of the following:

   a. When qualifying data-name-3 with data-name-4, data-name-4 must be the group item data-name that subordinates data-name-3.

   b. When qualifying data-name-3 with report-name-2, report-name-2 must be the report description entry report-name associated with data-name-3.

2. Specify qualifiers in hierarchy order from low-order qualifiers to high-order qualifiers moving from left to right.

F

Rules for Format 7

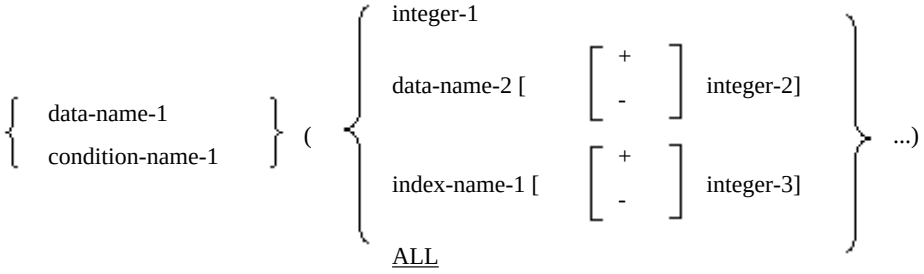
1. The EDIT-MODE, EDIT-OPTION, EDIT-OPTION2, EDIT-OPTION3, EDIT-COLOR, EDIT-STATUS, and EDIT-CURSOR special registers must be qualified before being referenced.
2. Identifier-1 must be a data item defined as having an item control field. See "NetCOBOL User's Guide" for more information on the item control field.
3. Identifier-1 must not be reference modified.
4. The necessary qualifiers and subscripts must be added to identifier-1.

1.4.2 Subscripting

Subscripting is used when referencing an individual element of a table. A subscript is added to the data-name or condition-name before the table elements are referenced.

Refer to "Subscripting" in the "COBOL Syntax Samples", for an example of the usage.

Format



Note

In the above format, data-name-1 and condition-name-1 are shown for clarity, they are not part of the subscript.

Syntax Rules

1. Data-name-1 must be a data item subordinate to a data item where an OCCURS clause has been specified or a similar data item.
2. The data item (conditional variable) associated with condition-name-1 must be a data item where an OCCURS clause was specified or a similar data item. When subscripting is required to reference the conditional variable, subscripting having an identical combination as the conditional variable must be added to the condition-name.
3. A table element must be subscripted before being referenced except in the following cases:
   - Subject in a SEARCH statement
   - REDEFINES clause
   - KEY IS phrase in an OCCURS clause

4. The number of subscripts must be the same as the number of OCCURS clauses written in the data description entry defining the table elements. When two or more subscripts are required, specify the subscripts in order from high-order dimensions to low-order dimensions in the table starting from left to right.
5. Index-name-1 must be the index-name specified in the OCCURS clause INDEXED BY phrase of the data description entry defining the table elements.
6. Data-name-2 must be an integer item.
7. Data-name-2 can be qualified.
8. The plus sign can be added to integer-1.
9. The subscript ALL can be used only when specifying a function argument. ALL cannot be written as the subscript for condition-name-1.
10. The SUM-COUNTER, LINE-COUNTER, and PAGE-COUNTER cannot be used as a subscript in the REPORT SECTION.
11. Data-name-1 and condition-name-1 can be qualified.

General Rules

1. One dimension corresponds to one OCCURS clause. The occurrence number indicates an element of the table in one dimension is being referenced. The maximum occurrence count specified in the OCCURS clause may be represented by n. In this case, the first table element in the dimension is represented by occurrence number 1 and the second table element is represented by occurrence number 2. The final table element in the dimension is represented by occurrence number n.
2. The maximum number of subscripts (the maximum number of dimensions) is 7.
3. When integer-1 or data-name-2 has been specified in a subscript, the value of integer-1 or data-name-2 represents the table element occurrence number.
4. When index-name-1 has been specified in a subscript, the value of index-name-1 represents the table element occurrence number.
5. The index-name-1 value must be initialized before being used as a subscript. The initial value of the index-name-1 can be set by a PERFORM statement containing a VARYING phrase, a SEARCH statement containing an ALL phrase, or a SET statement.
6. When integer-2 has been written, the subscript has one of two values. If the operator is "+", the value is obtained by adding the value of integer-2 to the value of data-name-2. If the operator is "-", the value is obtained by subtracting the value of integer-2 from the value of data-name-2.
7. When integer-3 is specified, the subscript has one of two values. If the operator is "+", the value is obtained by adding the value of integer-3 to the index-name-1 value. If the operator is "-", the value is obtained by subtracting the integer-3 value from the index-name-1 value.
8. When ALL has been written in the subscript, the program references all table elements in the dimension corresponding to the subscript.

1.4.3 Reference Modification

Reference modification references part of a data item or function value. It adds a reference modifier to the data-name or function-identifier.

Refer to "Reference Modification" in the "COBOL Syntax Samples", for an example of the usage.

Format 1

Reference modification of data item:

```
data-name-1 (high-order-end-character-position: [length])
```

Format 2

Reference modification of function value:

FUNCTION function-name-1 [({ argument-1 } ...)] (high-order-end-character-position: [length])

Note

In the above formats, "data-name-1" and "FUNCTION function-name-1[({argument-1} ...)] " are shown for clarity; they are not part of the reference modifier.

Syntax Rules

1. Data-name-1 must be an elementary item or group item.
2. The high-order-end-character-position and length must be an arithmetic expression.
3. Data-name-1 can be qualified or subscripted.
4. Function-name-1 must be an alphanumeric function.

F

   In [HP], [Win16], [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET], the name of a national function can be specified.

F

5. A strongly typed group item cannot be subjected to reference modification.
6. An elementary item subordinate to a strongly typed group item can be subjected to reference modification only when it is alphanumeric or national.

General Rules

1. Reference modification redefines part of the data item in data-name-1 or a temporary data item having a function value (the source data item for reference modification) as a unique data item (the data item resulting from reference modification).
2. The starting position in reference modification is specified in the "high-order-end-character-position" field. The high-order-end-character-position value must be an integer in the following range:

$$1 \leq \left(\begin{array}{l}\text{Value of}\\ \text{high-order-end-}\\ \text{character-position}\end{array}\right) \leq \left(\begin{array}{l}\text{number of character positions in}\\ \text{source data item for reference}\\ \text{modification}\end{array}\right)$$

3. Set the reference modification range in "length". The value of "length" must be an integer in the following range:

$$1 \leq (\text{length}) \leq \left(\begin{array}{l}\text{number of character Positions}\\ \text{for source data item for}\\ \text{reference modification}\end{array}\right) - \left(\begin{array}{l}\text{value of}\\ \text{high-order-end-}\\ \text{character-position}\end{array}\right) + 1$$

4. When "length" is omitted, the assumed length is computed as follows: the maximum length of the source data - high-order-end- character-position.
5. When a subscript is added to data-name-1, the reference modifier is evaluated immediately after the subscript evaluation. When ALL has been specified in the data-name-1 subscript, reference modifiers are applied to all table elements in the dimension corresponding to the subscript.
6. The data item resulting from reference modification is regarded as an elementary item having no JUSTIFIED clause.
7. After the function value has been determined, the reference modifier in Format 2 is applied to a temporary data item which contains the function value.

8. In Format 1 the data item resulting from reference modification is regarded as having the following category and class:

   - When the category of data-name-1 is alphabetic character, numeric character, numeric edited, alphanumeric character, or alphanumeric edited, the data item category and class resulting from reference modification are regarded as alphanumeric characters.
   - When the category of data-name-1 is National or National edited, the data item category and class resulting from reference modification are regarded as national language.
   - When the category of data-name-1 is Boolean, the data item category and class resulting from reference modification are regarded as alphanumeric characters.
   - When the category of data-name-1 is External floating, the data item category and class resulting from the reference modification are regarded as alphanumeric characters.
   - When data-name-1 is a group item, the data item category and class resulting from reference modification are regarded as alphanumeric characters.

9. In Format 2, the data item category and class resulting from reference modification are regarded as alphanumeric characters or national characters.

1.4.4 Pointer

A pointer is used for referencing the data-name or condition-name defined in the BASED-STORAGE SECTION. A pointer involves a pointer qualifier to the data-name or condition-name. The pointer qualifier indicates the data item address storage area defined in the BASED-STORAGE SECTION.

Refer to "Pointer" in the "COBOL Syntax Samples", for an example of the usage.

Format

```
[( ]...  { data-name-1                    }
         { data-name-2 ( { subscript } ... ) } ->
         { ADDR-function                  }

[ { data-name-1                    } ) ->  ] ...  { data-name-3     }
  { data-name-2 ( { subscript } ... ) }           { condition-name-1 }
```

In the above format, data-name-3 and condition-name-1 are shown for clarity, they are not part of the pointer qualifier. The sign "->" is a key word, but is not underlined to avoid confusion with other symbols.

Syntax Rules

1. Data-name-1 and data-name-2 must be pointer data items.
2. The symbol "->" is called the pointer qualifier symbol. A space can be written immediately before or after the pointer qualifier symbol. The pointer qualifier symbol must be contained on a single line.
3. When writing two or more pointer qualifier symbols, parentheses must enclose the portion preceding the second and subsequent pointer qualifier symbols.

```
PTR1->X
(PTR1->PTR2)->X
((PTR1->PTR2)->PTR3)->X
```

4. Data-name-1 and data-name-2 may be qualified.
5. Data-name-3 and condition-name-1 must be defined in the BASED-STORAGE SECTION.
6. Data-name-3 and condition-name-1 can be qualified or subscripted. Data-name-3 can be reference modified.

General Rules

1. When data-name-3 is a data item where the BASED ON clause has been omitted or is a data item subordinate to such a data item, a pointer qualifier must be added to data-name-3 before being referenced.
2. When the condition variable associated with condition-name-1 is a data item where the BASED ON clause was omitted or is a data item subordinate to such a data item, a pointer qualifier must be added to condition-name-1 before being referenced.
3. An implicit pointer qualifier can be defined by writing a BASED ON clause in the BASED-STORAGE SECTION data description entry. A pointer using an implicit pointer qualifier without writing a pointer qualifier is called an implicit pointer. A pointer that uses a pointer qualifier is called an explicit pointer.
4. A data item where a BASED ON clause has been specified or a data item subordinate to such a data item allows either an implicit or an explicit pointer qualifier to be added. When the conditional variable associated with the condition-name is a data item where a BASED ON clause has been specified or is a data item subordinate to such a data item, either an implicit or an explicit pointer qualifier can be added to the condition-name.

1.4.5 Identifier

A data-name made unique by the addition of

- a qualifier,
- a subscripting,
- a reference modification,
- F or a pointer

is called an "identifier". When creating a unique reference to a data-name, the data-name must be written in the identifier format.

Format

```
[pointer qualifier] data-name-1
[ { IN } data-name-2 ] ... [ { IN } { file-name-1 } ]
  { OF }                   { OF } { report-name-1 }
[({subscript} ...)] [reference modifier]
```

1. IN and OF are synonyms.
2. An identifier may be specified wherever "identifier-n" is indicated in a syntax diagram. A data-name where
   - a qualifier,
   - a subscript,
   - a reference modifier,
   - F or a pointer

   has been added to provide uniqueness of reference must be specified wherever "identifier-n" is indicated in a syntax diagram.
3. F A pointer qualifier can be written only when data-name-1 has been defined in the BASED-STORAGE SECTION.

1.4.6 Condition-name Uniqueness of Reference

A condition-name must be made unique by the addition of

- a qualifier,
- a subscript,

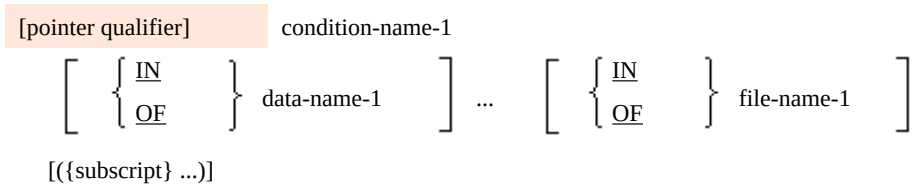
F - or a pointer

except where uniqueness of reference can be guaranteed by the rules for the scope of a name.

Refer to "Reference Modification" in the "COBOL Syntax Samples", for an example of the usage.

For details about how to use the examples, refer to "COBOL Syntax Samples", "Uniqueness of Reference of Condition Name".

Format



1. IN and OF are synonyms.
2. A condition-name may be specified wherever "condition-name-n" is indicated in a syntax diagram. A condition-name where
   - a qualifier,
   - a subscript,
F - or a pointer

   has been added to provide uniqueness of reference must be specified wherever "condition-name-n" is indicated in a syntax diagram.
F 3. A pointer qualifier can be written only when condition-name-1 is defined in the BASED-STORAGE SECTION.

1.4.7 Function-identifier

A function-identifier is a format used for a function. The function-identifier is treated as a temporary data item that contains a function value.

Format

FUNCTION function-name-1 [({ argument-1}...)][reference modifier]

1. Argument-1 must be an identifier, literal, or arithmetic expression.
2. A reference modifier can only be specified for :
   - an alphanumeric character function, or
F - a national function

   A reference modifier is used to provide for reference modification of function values.
3. The section titled "General Rules for Functions" of Chapter 6, the "Procedure Division" explains the locations where a function-identifier may be used.
4. The function argument is evaluated according to the position where argument-1 was specified.
5. A function-identifier or an expression including a function-identifier can be specified in argument-1. Function-name-1 can also be specified in argument-1.

1.5 Reference Format

The format used when writing a COBOL source program is called the "reference format". A COBOL source program must be written according to the reference format. When using the source text manipulation module, the COBOL library must also be written according to the reference format.

1.5.1 Configuration of a Source Code Line

The reference format is a rule stating which element is being placed in which position on a line. There are three types of reference formats:

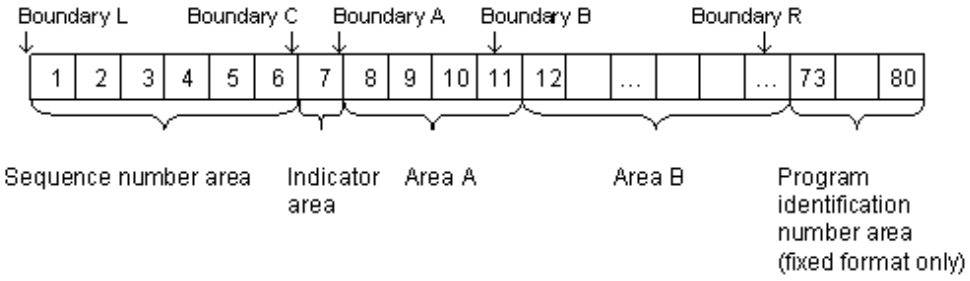
- Fixed format
- Variable format
- Free format

F

F

Free format is specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

The diagram below shows the configuration of a source code line when using either the fixed or variable reference format.



- Boundary L is immediately to the left of character position 1 of the line.
- Boundary C is immediately to the left of character position 7 of the line.
- Boundary A is immediately to the left of character position 8 of the line.
- Boundary B is immediately to the left of character position 12 of the line.
- Boundary R is one of the following positions:
  - In the fixed format, immediately to the left of character position 73 of the line.
  - In the variable format, immediately to the right of the last character position in the line. See "Appendix B, System Quantitative Restrictions" for more information on the last character position in the line.

Sequence Number Area

"Sequence number area" refers to the six character positions between boundary L and boundary C. It identifies a line in the source program. Any character belonging to the computer character set may be used in the sequence number area. The contents of the sequence number area need not be in a specific sequence or be unique.

Indicator Area

- "Indicator area" refers to the character position between boundary C and boundary A. An asterisk "*", slash "/", hyphen "-", or "D" can be specified in the indicator area.
- A line where an asterisk or slash has been specified in the indicator area is called a "comment line". The contents of area A and area B in a comment line are regarded as annotation.
- A hyphen in the indicator area indicates the first character in the line is joined to the last character of the previous line.
- A line where "D" has been written in the indicator area is called a "debugging line". A debugging line leaves debugging information in the source program.

Area A

"Area A" refers to the four character positions between boundary A and boundary B.

Area B

"Area B" refers to the area between boundary B and boundary R.

F

Program Identification Number Area

Only the fixed format has a program identification number area. "Program identification number area" refers to the eight character positions from the position immediately to the right of boundary R to character position 80.

1.5.2 Reference Format for Area A and Area B

Some COBOL words must start in area A and others in area B.

Words That Must Start in Area A

The following words must start in area A:

- Division header
- Section header
- Paragraph header
- Paragraph-name
- Level indicator (FD, SD or RD)
- Level-numbers 01 and 77
- The beginning (DECLARATIVES) and end (END DECLARATIVES) of a declarative
- END PROGRAM header

Words That Must Start in Area B

The following words must start in area B:

- Statements (except the COPY and REPLACE statements, which can start in area A)
- An entry (except entries starting with a level indicator or data description entries whose level-number is 01 or 77, which must originate in area A. Data description entries whose level-number is not 01 or 77 can also originate in area A.)
- Comment entry

Rules for Each Division

In addition to the rules stated above, the following rules also apply to the various divisions:

1. In the IDENTIFICATION DIVISION, a paragraph header, an entry or comment entry can be written on the same line. Two or more comment entries can be written on the same line. A single comment entry can span two or more lines.
2. In the ENVIRONMENT DIVISION, a paragraph header and an entry can be written in the same line. Two or more entries can be written on the same line. A single entry can span two or more lines.
3. In the DATA DIVISION, two or more entries can be written on the same line. A single entry can span two or more lines. When the level-number is a single digit, it can be written with a single numeric character. For clarity, the starting position of the level-number can be changed as additional levels are added.
4. In the PROCEDURE DIVISION, a paragraph-name and sentence can be written on the same line. Two or more sentences can be written on the same line. A single sentence can span two or more lines.

1.5.3 Blank Line

A line whose indicator area (areas A and B) is all spaces is called a "blank line". A blank line can be written anywhere in a source program or in library text.

1.5.4 Comment Line

A line where an asterisk (*) or slash (/) has been written in the indicator area is called a "comment line". A comment line can be written before the IDENTIFICATION DIVISION header, on any line after the IDENTIFICATION DIVISION header, or on any line of library text.

Any characters belonging to the computer character set can be written in area A and B of a comment line. A comment line is regarded a comment, and is printed in the source code listing produced by the compiler.

In a source code compiler listing, a comment line is printed immediately below the previous line. A comment line where a slash has been written in the indicator area is printed after a form feed (on a new page).

1.5.5 Continuation of Lines

A character-string consisting of sentences, entries, and clauses can be continued from one line to area B in the next line. The line to where a character-string is continued is called a "continuation line". The line where a character-string was continued is called a "continued line".

Refer to "Continuation of a Line" in the "COBOL Syntax Samples", for an example of the usage.

The following character-strings can be continued to a continuation line.

- COBOL words excluding National user-defined words
- Literal
- PICTURE Character-string

When continuing character strings to a continuation line, the indicator area in the continuation line must contain a hyphen (-). Area A in the continuation line must be spaces. A hyphen in the indicator area indicates the first character not being a space in area B of the continuation line continues on from the last character not being a space in the continued line (excluding a comment line and blank line).

Continuation of a

- nonnumeric literal,
- F hexadecimal nonnumeric literal,
- national nonnumeric literal,
- or Boolean literal

to the continuation line must also conform to the following rules:

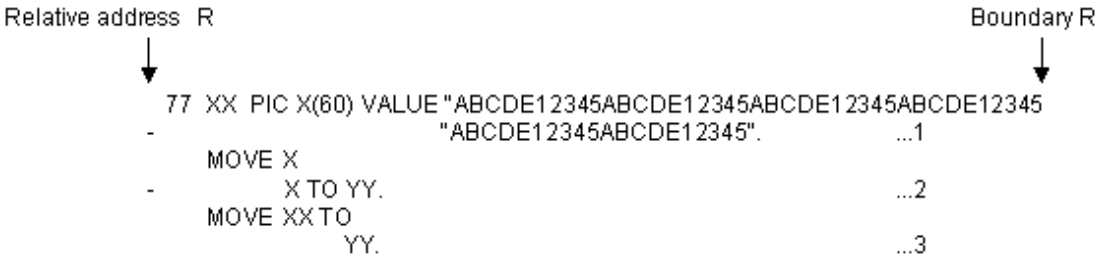
1. The first character not being a space in area B of the continuation line must be a quotation mark. The continuation of the literal must start at the character position immediately to the right of the quotation mark.
2. Any spaces to the end of area B in the continued line are regarded as part of the literal. The literal must be written so the last character in area B of the continued line continues to the character immediately following the first quotation mark in the continuation line.

To continue a line when the character-string fits into one line and is not to be continued to the next line, a hyphen must not be written in the indicator area. When no hyphen exists in the indicator area, the position immediately preceding a character not being a space in the continuation line is regarded as a space.

A separator (i.e., one of the following character-strings) must be written on a single line:

- The == pseudo-text delimiter
- F A separator indicating the start of a literal value (X", N", NC", NX" or B")
- The -> pointer qualification symbol

The diagram below shows examples of continuing lines.



1. The nonnumeric literal is continued.
2. The user-defined word XX is continued.
3. No word or literal is split - so no continuation hyphen is needed and none should be entered (otherwise the words TO and YY would be combined).

1.5.6 Debugging Line

A line where a D has been written in the indicator area is called a "debugging line". A debugging line leaves debugging information in the source program. The debugging line can be written after the OBJECT-COMPUTER paragraph in the IDENTIFICATION DIVISION.

Areas A and B of the debugging line must conform to legal COBOL syntax. When areas A and B of the debugging line are spaces, the debugging line is regarded as a blank line.

Inclusion of the WITH DEBUGGING MODE clause in the SOURCE-COMPUTER paragraph of the ENVIRONMENT DIVISION specifies that the debugging line is to be compiled. That is, the debugging line is compiled only when the WITH DEBUGGING MODE clause is written. When this clause was omitted, the debugging line is regarded as a comment line.

At compilation, the compiler determines whether the WITH DEBUGGING MODE clause was specified after processing the COPY and / or REPLACE statements.

F

1.5.7 In-line Comment

To write a comment on the same line as a statement or entry, use an in-line comment. The in-line comment must begin after the rightmost character position of the statement or entry. An in-line comment is the portion starting with the two characters "*>" and ending at the character position immediately to the left of boundary R on the same line. An in-line comment can be written in the IDENTIFICATION DIVISION header line or any line thereafter, or in any library text line.

A separator space must immediately precede the "*>" symbol.

Any character belonging to the computer character set can be written after the "*>" symbol. The in-line comment is regarded as a comment.

F

1.5.8 Reference Format for Free Format

This section explains the free format reference format.

Refer to "Free Format" in the "COBOL Syntax Samples", for an example of the usage.

However, an obsolete language element cannot be used in the free format reference format. Free format is specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

1.5.8.1 Configuration of a Line

In the free format reference format, the source program text can be written anywhere in a line with the exception of the particular rules for comment, debugging line, and continuation of lines. The number of character positions in each line can be from 0 to 251. (A national character occupies two or more character positions.)

1.5.8.2 Blank Line

A line composed of all spaces is called a "blank line". A blank line can be written anywhere in a source program or library text.

1.5.8.3 Comment Lines

In the free format reference format, only in-line comments are allowed. See the "In-line Comment" topic above for the rules for in-line comments.

1.5.8.4 Continuation of Lines

A character-string consisting of sentences, entries, and clauses can be continued from one line to the next line. This subsequent line is referred to as a "continuation line" while the previous line is referred to as a "continued line."

Continuation of nonnumeric literal, Boolean literal, or national nonnumeric literal can be written separately to the continuation line. When these literals have been separated at the end of a line, the separated literal must be ended in the quotation mark immediately following the hyphen. Zero or one or more separator space must be continued after the hyphen.

The first character not being a space in the continuation line must be a quotation mark. The continuation of the literal must start at the character position immediately to the right of the quotation mark. At this point, one or more characters of the literal must be written both in the continued line and in the continuation line.

A separator composed of two or more characters must be written in the same line. A pair of quotation marks, which indicates a quotation mark in a literal, must be described in the same line. A comment line and a blank line can be written anywhere between lines including part of a literal.

1.5.8.5 Debugging Line

Combining three COBOL characters ">>D" immediately followed by a single-byte space indicates a debugging indicator. A line with zero or one or more blanks before the debugging indicator is called "debugging line". The debugging line leaves debugging information in the source program. The debugging line can be written after the OBJECT-COMPUTER paragraph in the IDENTIFICATION DIVISION. The debugging line ends at the end of the line.

The debugging line must conform to the rules for COBOL syntax. If zero or one or more blanks have been written in the debugging line, the debugging line is regarded as a blank line.

Inclusion of the WITH DEBUGGING MODE clause in the SOURCE-COMPUTER paragraph of the ENVIRONMENT DIVISION specifies the debugging line is to be compiled. Therefore, the debugging line is compiled only when the WITH DEBUGGING MODE clause is written. When this clause is omitted, the debugging line is regarded as a comment line.

At compilation, the compiler determines whether the WITH DEBUGGING MODE clause has been specified after processing the COPY and REPLACE statements.

1.6 Program Configuration

A program consists of the following four divisions and an END PROGRAM header:

- IDENTIFICATION DIVISION
- ENVIRONMENT DIVISION
- DATA DIVISION
- PROCEDURE DIVISION

A program begins with the IDENTIFICATION DIVISION and ends with the END PROGRAM header or the last line of the program.

Other programs can be written between the start and end of the program. A program contained within another program is called a "nested program". A nested program can be written provided the inter-program communication module is used.

The scope between the outermost start and end of a program is called the "compilation unit". One or more nested programs can be written in one compilation unit. The configuration where one program contains another program is called "program nesting".

A nested program can itself contain a nested program. When one program (program A) contains another program (program B) immediately within it and program B also contains another program (program C) immediately within it, program B is said to be "contained directly" within program A and program C is said to be "contained indirectly" within program A.

The formats of programs in a compilation unit are shown below:

Format 1

Compilation unit:

IDENTIFICATION DIVISION

[ENVIRONMENT DIVISION]

[DATA DIVISION]

PROCEDURE DIVISION]

[Nested program] ...

[END PROGRAM header]

Format 2

Nested program:

IDENTIFICATION DIVISION

[ENVIRONMENT DIVISION]

[DATA DIVISION]

[PROCEDURE DIVISION]

[Nested program] ...

END PROGRAM header

1. When writing nested programs in a compilation unit, an END PROGRAM header must be written for each program. For example, when program A contains program B and program B in turn contains program C, an END PROGRAM header must be written separately for each of the programs A, B, and C.
2. The outermost program in a compilation unit must end with an END PROGRAM header. However, this header can be omitted from the last program in a series of compilation unit programs.
3. A division header indicates the beginning of each division. The end of each division is indicated by one of the following:
    a. The next division header in the same program
    b. The IDENTIFICATION DIVISION header indicating the start of a nested program
    c. The END PROGRAM header
    d. A physical position where there is no other line.

F

1.6.1 Returning Result from a Program

If the RETURNING phrase is given in the Procedure Division header of a program definition, the result of a program will be stored in the data item indicated in the RETURNING phrase, when control is returned to the calling program. This result is stored in the identifier denoted in the RETURNING phrase of the CALL statement in the calling program.

F

1.7 Operation Mode

Operation mode defines the maximum number of digits that can be used for a numeric item, an edited numeric item and a numeric literal. Operation mode also defines the method by which arithmetic expressions are processed.

There are two operation modes:

18-digit compatibility operation mode

18-digit mode is used for compatibility between systems and between Version Levels.

31-digit extension operation mode

Up to 31 digits can be used for numeric items, edited numeric items and numeric literals in this mode. 31-digit extension mode is used for functions that have been enhanced to use more than 18 digits. 31-digit extension operation mode is available in [Winx64] and [Linux64].

Point

In [Winx64] and [Linux64], the operation mode can be selected via compiler option ARITHMETIC.

Differences between the operation modes

The table below shows the differences between 18-digit compatibility operation mode and 31-digit extension operation mode.

Item		18-digit compatibility operation mode	31-digit extension operation mode
System	[Winx64] [Linux64]	available	available
	Others	available	not available
Number of digits that can be used in a numeric item, an edited numeric item, or a numeric literal		18 digits or fewer	31 digits or fewer
Precision of the intermediate result of a fixed point arithmetic operation		30 digits or fewer	38 digits or fewer
Number of digits when the intermediate result of an arithmetic expression is floating point		- Each arithmetic operation is considered separately. - When at least one operand of an arithmetic operation is a floating-point data item or a floating-point literal, the intermediate result will be floating-point.	- All elements of each arithmetic statement, each conditional expression, and each arithmetic expression that it contains are considered. - The floating point attribute is applied to the entire arithmetic statement, conditional expression, or arithmetic expression intermediate result if the statement or expression includes a floating point in one of its elements.

Chapter 2 COBOL Modules

COBOL has the following ten modules:

- Nucleus module
- Input-output module (sequential, relative and indexed)
- Inter-program communication module
- Sort-merge module
- Source text manipulation module
- F - Presentation file module
- Intrinsic function module
- F - Screen handling module
- Command line argument and environment variable operation module
- Report writer module

This chapter gives an overview of each module. With the exception of "Nucleus," the module to which a COBOL verb belongs is indicated in the text, as an aid to association between the statement and the COBOL function.

2.1 Nucleus

The nucleus is an elementary module for converting, comparing, and performing operations on data. The nucleus consists of the following modules:

- Data transcription: MOVE statement
- Arithmetic operation: COMPUTE statement, ADD statement, DIVIDE statement, MULTIPLY statement, and SUBTRACT statement
- Optional processing: IF statement, EVALUATE statement, and CONTINUE statement
- Branch: GO TO statement and ALTER statement
- Repeat processing: PERFORM statement, EXIT statement, and EXIT PERFORM statement
- Table handling: SEARCH statement and SET statement
- Initialization of data item: INITIALIZE statement
- Character-string handling: INSPECT statement, STRING statement, and UNSTRING statement
- Simple input-output: ACCEPT statement and DISPLAY statement
- End of program execution: STOP RUN statement
- F - Pointer handling: No statement
- Floating-point handling: No statement

Include the following statements as required when using the nucleus in a program.

Program Description	Main Facility
IDENTIFICATION DIVISION	Specifies the name of the program
ENVIRONMENT DIVISION	
CONFIGURATION SECTION	
SOURCE-COMPUTER paragraph	Specifies the debugging mode
OBJECT-COMPUTER paragraph	Specifies the collating sequence of characters

Program Description	Main Facility	
SPECIAL-NAMES paragraph	Defines the mnemonic-name, alphabet-name, symbolic-character, class-name, currency symbol, and decimal point	
	Defines the symbolic-constant ,positioning unit name , and print mode name	
DATA DIVISION		
BASED-STORAGE SECTION		
77-level description entry and record description entry	Defines the data item to which a pointer is to be added and referenced	
WORKING-STORAGE SECTION		
77-level description entry and record description entry	Defines the data item	
LOCAL-STORAGE SECTION		
77-level description entry and record description entry	[Winx64][Linux64][.NET] Defines the automatic item	
CONSTANT SECTION		
77-level description entry and record description entry	Defines a data item as a constant	
Program Description	Main Facility	
PROCEDURE DIVISION		
Procedure Division statements		
MOVE	Moves and transcribes data	
COMPUTE	Performs an arithmetic operation	
ADD	Performs addition	Arithmetic operations
DIVIDE	Performs division	
MULTIPLY	Performs multiplication	
SUBTRACT	Performs subtraction	
IF	Distributes processing according to the truth value of a condition	Optional processing
EVALUATE	Distributes processing according to the truth value of several conditions	
CONTINUE	No-operation statement	
GO TO	Shifts control to another location in the PROCEDURE DIVISION	Branch
ALTER	Changes the branch destination of a GO TO statement	
PERFORM	Repeats a procedure	Repeat processing
EXIT	Specifies the common exit for outer PERFORM statements	
EXIT PERFORM	Specifies the common exit for inner PERFORM statements	
SEARCH	Retrieves a table element	Table handling
SET	Sets the coordinates of a table element	
INITIALIZE	Initializes a data item	

Program Description			Main Facility	
		INSPECT	Verifies a character-string	Character-
		STRING	Links a character-string	string
		UNSTRING	Decomposes a character-string	handling
		ACCEPT	Inputs data from the hardware unit	Simple
		DISPLAY	Displays data on the hardware unit	input-output
		STOP RUN	Ends a program	
END PROGRAM header			Specifies the end of a program	

2.1.1 Transcription and Movement of Data

The MOVE statement copies, or transcribes, data to one or more data locations. It also tailors the data according to the attributes of the receiving item, providing for data alignment, data type conversion, and editing.

Examples of the MOVE statement are given below.

DATA DIVISION

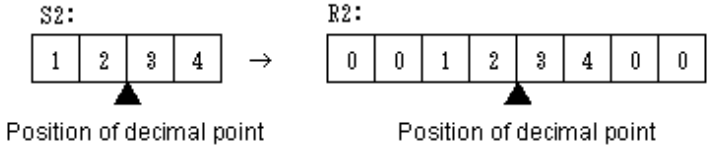
```
77 S1 PIC X(4).
77 R1 PIC X(6).
77 S2 PIC 9(2)V99.
77 R2 PIC 9(4)V9(4).
77 S3 PIC 9(6).
77 R3 PIC ZZZ,ZZ9.
```

PROCEDURE DIVISION

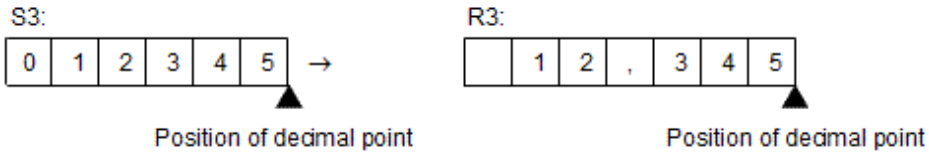
```
        MOVE S1 TO R1.              *>...1
        MOVE S2 TO R2.              *>...2
        MOVE S3 TO R3.              *>...3
```

MOVE statement 1 transfers and transcribes S1 to R1. It positions the data from the left end of the receiving item and fills the unused positions with spaces.

MOVE statement 2 transfers and transcribes S2 to R2 with decimal alignment, and pads the receiving area with zeros.



MOVE statement 3 transfers and transcribes S3 to R3 with decimal alignment and editing. It replaces the leading zero with a space and inserts a comma.



2.1.2 Arithmetic Operations

Use the ADD statement to perform addition, the SUBTRACT statement to perform subtraction, the MULTIPLY statement to perform multiplication, the DIVIDE statement to perform division, and the COMPUTE statement to perform various arithmetic operations.

F

The COMPUTE statement can also be used to perform a Boolean operation.

The ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPUTE statements are generically called "arithmetic statements." An arithmetic statement can round off the result of an arithmetic operation and also perform an overflow check. To round off a value, use the ROUNDED phrase. To run an overflow check, use the ON SIZE ERROR phrase.

Examples of arithmetic statements are given below.

```
ADD C TO A.                          *>...1
DIVIDE C BY D GIVING A ROUNDED.      *>...2
COMPUTE A = C / D + E * 100.         *>...3
COMPUTE Z = X AND Y.                 *>...4
```

1. Stores the result of A + C in A.
2. Rounds off the quotient of C/D and stores the result in A.
3. Stores the result of C/D + E * 100 in A.
4. Stores the Boolean product of X and Y in Z.

2.1.3 Optional Processing and Branching

The IF and EVALUATE statements are used for conditional processing. The IF statement checks a condition and selects the next executable statement depending on the truth value of the condition. The EVALUATE statement checks a condition and selects one process from among several.

Use the GO TO statement to branch unconditionally from one location to another location.

Example of the IF Statement

In an IF statement, use the THEN phrase to indicate processing if a condition is true and the ELSE phrase to indicate processing if the condition is false.

An example of the IF statement is given below.

```
      IF X = Y THEN
        MOVE 0 TO Z      *>...1
      ELSE
        GO TO P1         *>...2
      END-IF.
      *>  :
P1.
*>      :
```

1. If X=Y is true, the MOVE statement is executed.
2. If X=Y is false, the GO TO statement is executed and the program branches to paragraph P1.

Example of the EVALUATE Statement

In an EVALUATE statement, use the WHEN phrase to indicate what processing should be executed. The identifier written immediately after EVALUATE and the identifier written immediately after WHEN are compared. If the conditions are true, the statement written in the WHEN phrase is executed. An example of the EVALUATE statement is given below.

```
       EVALUATE MARKS
         WHEN 85 THRU 100 MOVE "A" TO RESULT        *>...1
         WHEN 70 THRU  84 MOVE "B" TO RESULT        *>...2
         WHEN 55 THRU  69 MOVE "C" TO RESULT        *>...3
         WHEN OTHER       MOVE "D" TO RESULT        *>...4
       END-EVALUATE.
```

1. When the value of MARKS is between 85 and 100, the statement MOVE "A" TO RESULT is executed.
2. When the value of MARKS is between 70 and 84, the statement MOVE "B" TO RESULT is executed.
3. When the value of MARKS is between 55 and 69, the statement MOVE "C" TO RESULT is executed.
4. When MARKS does not meet the conditions in (1), (2) or (3), the statement MOVE "D" TO RESULT is executed.

The EVALUATE statements above can be written using condition-names as follows.

[DATA DIVISION]

```
 01 MARKS PIC 9(3).
    88 RESULT-A VALUE 85 THRU 100.
    88 RESULT-B VALUE 70 THRU 84.
    88 RESULT-C VALUE 55 THRU 69.
 01 RESULT PIC X.
```

[PROCEDURE DIVISION]

```
       EVALUATE TRUE
             WHEN RESULT-A MOVE "A" TO RESULT
             WHEN RESULT-B MOVE "B" TO RESULT
             WHEN RESULT-C MOVE "C" TO RESULT
             WHEN OTHER    MOVE "D" TO RESULT
       END-EVALUATE.
```

2.1.4 Repetitive Processing

Use the PERFORM statement for repetitive processing of one or more statements (or procedures). There are two kinds of PERFORM statements:

- Outer PERFORM statement
- Inner PERFORM statement

Outer PERFORM Statement

The outer PERFORM statement specifies the procedure(s) to be repetitively executed.

An example of an outer PERFORM statement is given below.

```
      MOVE 0 TO TBL-SUM.
      PERFORM P1 VARYING I FROM 1 BY 1 UNTIL I > 5
                  AFTER J FROM 1 BY 1 UNTIL J > 10    *>...1
      *>    :
      PERFORM P2 THRU PX.                             *>...2
      *>    :
P1.                                          *> -+ Scope of PERFORM statement 1
      ADD TBL-X (I, J) TO TBL-SUM.           *> -+
P2.                                          *>    -+
                                             *>     | Scope of PERFORM statement 2
```

```
PX.                                          *>      |
     EXIT.                                   *>    -+
```

1. The PERFORM statement repeats the statement in paragraph-named P1. It sets the initial value of I to 1 and increments the value by 1 each time the statement in paragraph P1 is run. The PERFORM statement repeats this process until the value of I exceeds 5.
2. The PERFORM statement executes the statements between paragraph-name P2 and PX only once. The EXIT statement indicates the end of the paragraph.

Inner PERFORM Statement

When using the inner PERFORM statement, write the statement(s) to be repeated after the PERFORM statement.

An example of the inner PERFORM statement is given below.

```
       MOVE 0 TO TBL-SUM.
       PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5            *>...1
         PERFORM VARYING J FROM 1 BY 1 UNTIL J > 10         *>...2
           ADD TBL-X (I, J) TO TBL-SUM
         END-PERFORM
       END-PERFORM.
```

1. The PERFORM statement repetitively executes the statement written after the UNTIL phrase. It sets the initial value of I to 1 and increments the value by 1 each time the following statement is executed. When the value of I exceeds 5, the PERFORM statement transfers control to the statement following the outer END-PERFORM.
2. The PERFORM statement repetitively executes the ADD statement written after the UNTIL phrase. It sets the initial value of J to 1 and increments the value by 1 each time the ADD statement is executed. When the value of J exceeds 10, control is returned to PERFORM statement 1.

2.1.5 Table Handling

When defining multiple data items having the same attributes, the data items can be collectively defined in a table. To define a table, write the OCCURS clause in the data description entry. An element in the table is called a "table element." A table element is subscripted before being referenced.

Occurrence Count for Tables

The occurrence count (number of elements) for a table can be fixed or variable. When the occurrence count is fixed, the DEPENDING ON phrase in the OCCURS clause must be omitted. When the occurrence count is variable, use the DEPENDING ON phrase in the OCCURS clause. Specify the minimum value and maximum value for the occurrence count in the OCCURS clause. The actual occurrence count of the table is specified in the DEPENDING ON phrase. The data item specified in the DEPENDING ON phrase is called a "variable occurrence data item."

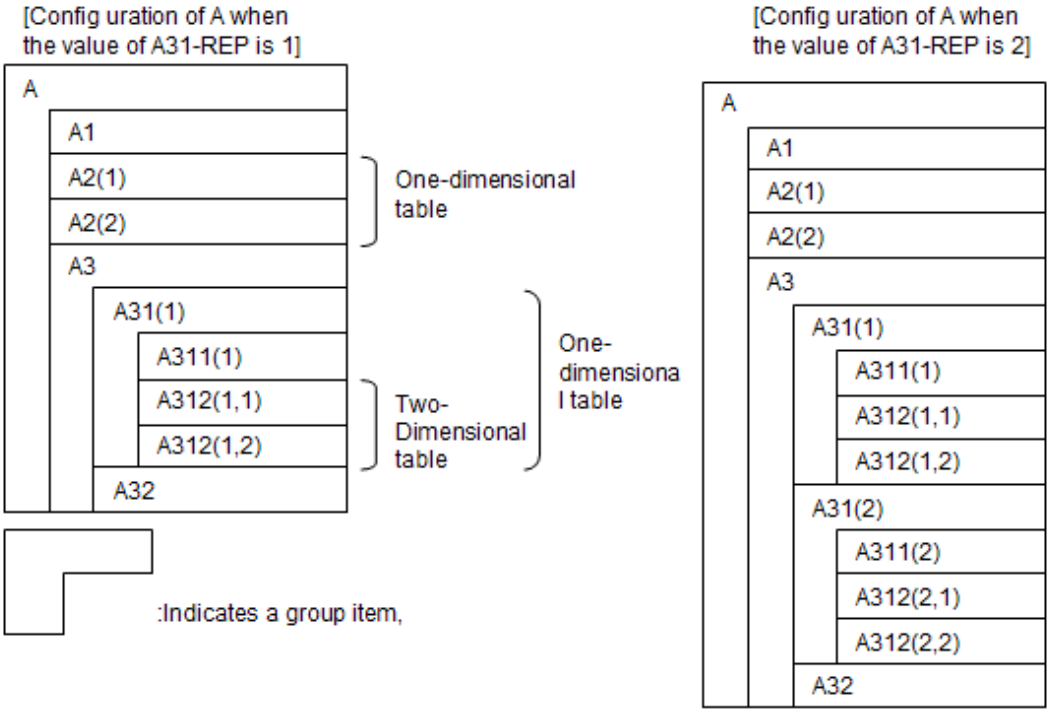
The variable occurrence data item in a record can be followed by another data item. The area allocated after the variable occurrence data item in a record is called the "variable position in the record."

An example of defining a table is given below.

```
01  A.
  02  A1 PIC X.
  02  A2 PIC X(2) OCCURS 2.                          *>...1
  02  A3.
    03  A31 OCCURS 1 TO 3 DEPENDING ON A31-REP.     *>...2
      04  A311 PIC X(4).
      04  A312 PIC X(5) OCCURS 2.                    *>...3
    03  A32 PIC X(3).                               *>...4
01  A31-REP PIC 9.                                  *>...5
```

1. Table with fixed occurrence count
2. Table with variable occurrence count

3. Table with fixed occurrence count
4. Variable position in record
5. Data item for setting occurrence count



The values (n) and (n, m) are subscripts. In the subscript A312 (n, m), n corresponds to the OCCURS clause written in data description entry A31 and m corresponds to the OCCURS clause written in data description entry A312.

Reference Area of Variable Occurrence Data Item

In the following explanation, X represents the data-name written in the DEPENDING phrase of an OCCURS clause and n represents the value of X at the start of processing. A statement referring to a group item having a subordinate variable occurrence data item processes the following areas within the variable occurrence data item area.

1. When the group item does not have a subordinate X, the first to the nth table elements are processed.
2. When the group item has a subordinate data-name-1, the first to the nth table elements are processed when using the group item as a sending item. When using the group item as a receiving item, the maximum value for the occurrence count is processed.

Index Name and Index Data Item

An integer, integer item, or index-name can be used as a subscript to a table element. When an integer or integer item is written as a subscript, its value represents the occurrence number of the table element. When an index-name is written as a subscript, its value represents the value for identifying the occurrence number of the table element.

The index-name is specified in the INDEXED BY phrase in the OCCURS clause. The compiler will create the index named in the INDEXED BY phrase. When subscripting the table, the index-name specified in the table INDEXED BY phrase must be used. The value of index-name is initialized with the SET statement.

The SET statement can also be used to set the value of a data item defined with the USAGE IS INDEX phrase. An index data item is a data item that can be used for storing the value of index-name.

An example of the application of the index-name and index data item is given below.

```
77 B USAGE INDEX.
01 C.
    02 C1 OCCURS 3 INDEXED BY IDX1.
```

```
         03 C11 PIC X(2).
         03 C12 PIC X(3) OCCURS 3 INDEXED BY IDX2.
  01  A PIC X(3).
PROCEDURE DIVISION.
       SET IDX1 TO 1.                                    *>...1
       SET B TO IDX1.                                    *>...2
       SET IDX2 TO 2.                                    *>...3
       MOVE C12(IDX1, IDX2) TO  A.                       *>...4
```

Defines an index data item named B,
Defines an index-name IDX1.
Defines an index-name IDX2,

1. Sets the value corresponding to occurrence number 1 in index-name IDX1 in table C1.
2. Saves the value of index-name IDX1 in index data item B.
3. Sets the value corresponding to occurrence number 2 in index-name IDX2 in table C12.
4. Adds the subscripts IDX1 and IDX2 to the table elements in table C12 and references it.

Retrieving Elements From a Table

Use the SEARCH statement to retrieve a table element meeting certain conditions. If the table element values are already in ascending or descending order, a SEARCH statement containing the ALL phrase can be used.

The following is an example of the SEARCH statement with the ALL phrase:

```
   01 TBL-VALUE.                                     *> -+...1
     02 FILLER .                                     *>  |
       03 FILLER PIC X(02) VALUE "01".               *>  |
       03 FILLER PIC X(20) VALUE "APPLE".            *>  |
     02 FILLER.                                      *>  |
       03 FILLER PIC X(02) VALUE "10".               *>  |
       03 FILLER PIC X(20) VALUE "ORANGE".           *>  |
     02 FILLER.                                      *>  |
       03 FILLER PIC X(02) VALUE "12".               *>  |
       03 FILLER PIC X(20) VALUE "PEACH".            *> -+
   01 TBL-REF REDEFINES TBL-VALUE.                   *>         -+...2
     02 TBL-DATA OCCURS 3 TIMES ASCENDING            *>          |
                KEY IS G-CODE INDEXED BY IDX.       *>...3      |
       03 G-CODE PIC X(02).                          *> -+...4   |
       03 G-NAME PIC X(20).                          *> -+      -+
PROCEDURE DIVISION.
     SEARCH ALL TBL-DATA                            *> -+...5
       AT END DISPLAY "ERROR"                       *>  |
         WHEN G-CODE (IDX) = "10"                   *>  |
           DISPLAY G-NAME (IDX)                     *>  |
     END-SEARCH.                                    *> -+
```

1. Defines the value of the table TBL-VALUE.
2. Defines the table TBL-REF. TBL-REF occupies the same area as TBL-VALUE.
3. Defines the table TBL-DATA. Set the occurrence count to 3, associate the index-name IDX with TBL-DATA, and use the ASCENDING KEY phrase to specify that the values of G-CODE are arranged in ascending order.
4. Defines G-CODE and G-NAME as the table elements subordinate to TBL-DATA.
5. When the program retrieves and finds a table element meeting the conditions written in the WHEN phrase, it executes the unconditional statement (DISPLAY statement) following the conditions in the WHEN phrase. When the program does not find such a table element, it executes the unconditional statement (DISPLAY statement) in the AT END phrase.

2.1.6 Initialization of Data Items

Use the INITIALIZE statement to initialize a data item.

An example of an INITIALIZE statement is given below.

```
    01 GRP1.
        02 A PIC X(8).
        02 B PIC S9(4) PACKED-DECIMAL.
        02 C PIC N(4).
PROCEDURE DIVISION.
        INITIALIZE GRP1.                              *>...1
        INITIALIZE GRP1 REPLACING NUMERIC DATA BY 1  *>...2
```

1. Initializes all of the elementary items subordinate to GRP1. A is initialized to a space, B is initialized to 0, and a National space placed in data item C.
2. Initializes only numeric data items in the group item named GRP1. Data item B initialized with the value 1. Data items A and C are left uninitialized.

2.1.7 Handling Character Strings

Use the INSPECT statement to count the number of occurrences of particular character string in a data item. You can also use the INSPECT statement to replace characters in a character string. The STRING statement is used to combine multiple character strings into another character string. The UNSTRING statement is used to break a character string into one or more character strings.

Example of the INSPECT Statement

An example of the INSPECT statement is given below.

```
    77 A PIC X(10) VALUE "AB*DE*FGH*".
    77 B PIC 99.
PROCEDURE DIVISION.
        INSPECT A TALLYING B FOR ALL "*".          *>...1
```

Inspects the contents of A, and counts the number of occurrences of "*". If the value of B prior to execution of the INSPECT statement is zero, it is initialized to the value 3 after the INSPECT statement is executed.

Example of the STRING Statement

An example of the STRING statement is given below.

```
    77 A PIC X(5) VALUE "ABCDE".
    77 B PIC X(3) VALUE "123".
    77 C PIC X(2) VALUE "+-".
    77 D PIC X(10).
PROCEDURE DIVISION.
        STRING       A DELIMITED BY SIZE
                     B DELIMITED BY SIZE
                     C DELIMITED BY SIZE
                     INTO D.                        *>...1
```

Combines the contents of A, B, and C in that order, and stores them in D. The contents of D after the STRING statement execution are "ABCDE123+-".

Example of the UNSTRING Statement

An example of the UNSTRING statement is given below.

```
    77 A PIC X(5).
    77 B PIC X(3).
    77 C PIC X(2).
    77 D PIC X(10) VALUE "ABCDE123+-".
PROCEDURE DIVISION.
        UNSTRING D INTO A B C.                      *>...1
```

The contents of D are broken apart, based on the size of the receiving items, and stored in data items A, B, and C, in the order given by the INTO clause of the UNSTRING statement. After the UNSTRING statement is executed, the contents of data items A, B and C are "ABCDE", "123" and "+-" respectively.

2.1.8 Simple Input-Output

Use the ACCEPT statement to retrieve input from a hardware device, and the DISPLAY statement to output data to a hardware device. The ACCEPT and DISPLAY statements are not used for file input or output. The ACCEPT statement can also be used to retrieve the date and time.

An example of the ACCEPT and DISPLAY statement is given below.

```
        DISPLAY "ERROR".                                    *> ...1
        ACCEPT X.                                           *> ...2
        ACCEPT C-TIME FROM TIME.                            *> ...3
```

1. Displays the word "ERROR" on the computer's console.
2. Allows the computer user to input data from the keyboard and stores the data in data item X.
3. Stores the current time in C-TIME.

2.1.9 Terminating a Program

Use the STOP RUN statement to terminate a program. This statement causes termination of a run unit.

F

2.1.10 Pointer Handling

Use pointers for referencing and manipulating the address of a data item.

The address of a data item is stored in a pointer data item by using the ADDR function. A pointer data item is defined by using the USAGE IS POINTER clause when defining the item. You define the structure of the data item that you want to reference in the BASED-STORAGE or the WORKING-STORAGE sections of the DATA DIVISION.

Pointer references can be explicit or implicit. An explicit pointer reference is specified by defining the data item in the BASED-STORAGE section and using the BASED ON phrase. An implicit pointer reference is indicated by using the pointer qualifier (the characters "->").

An example of handling a pointer is given below.

```
BASED-STORAGE SECTION.
 01  ITEM1   PIC X BASED ON P1.                          *>...1
 01  ITEM2   PIC 9(4).
 WORKING-STORAGE SECTION.
 01  P1      USAGE POINTER.                              *>...2
 01  ITEM3   PIC X.                                      *>...3
 LINKAGE SECTION.
 01  P2      USAGE POINTER.                              *>...4
 PROCEDURE DIVISION USING P2.
     MOVE FUNCTION ADDR(ITEM3) TO P1.                    *>...5
     MOVE "A" TO ITEM1.                                  *>...6
```

```
*>[Allocate memory and store the memory address in P2]                  ...7
     MOVE 1234 to P2->ITEM2.                                          *>...8
```

1. This instructs the compiler that when the pointer P1 is initialized with a valid address, that address can be referenced by the name ITEM1. Note that ITEM1 does not exist in memory until pointer P1 is initialized with an address.
2. Pointer P1 is defined.
3. ITEM3 is defined.
4. Pointer P2 is defined.
5. The address of ITEM3 is moved to the pointer P1. At this point in time, ITEM1 and ITEM3 have the same address in memory.
6. Now that pointer P1 has a valid address stored in it, ITEM1 can now be referenced. When the MOVE statement is executed, both ITEM1 and ITEM3 contain the character "A".
7. Use an operating system function to dynamically allocate four bytes of memory and store the address in pointer P2.
8. Associate the address in P2 with data item ITEM2, and move the value "1234" to ITEM2 (an implicit pointer reference).

F

2.1.11 Handling Floating-Point Data Items

A floating-point data item is numeric data item expressed in the format "mantissa*(10**exponent)." There are two types of floating-point data items:

- Floating-point data item
- Floating-point literal

In [Win16], [Win32], [Winx64], and [.NET], a single-precision floating-point literal can be 0 or have an absolute value greater than approximately 1.18 x 10 to the _38th power and less than about 3.4 x 10 to the +38th power. A double-precision floating-point literal can be 0 or have an absolute value greater than approximately 2.23 x 10 to the _308th power and less than about 1.79 x 10 to the +308th power. In [Linux], [LinuxIPF] and [Linux64], a floating-point literal can be 0 or have an absolute value greater than approximately 1.18 x 10 to the _38th power and less than about 3.4 x 10 to the +38th power. In [Solaris], [UXP/DS], and [HP], a floating-point literal can be 0 or have an absolute value greater than approximately 1.18 x 10 to the _38th power and less than about 7.2 x 10 to the +75th power.

A floating-point data item can be written wherever "data-name" or "identifier" is indicated below. A floating-point literal can be written wherever "literal" is indicated below.

ENVIRONMENT DIVISION

- Literal in a SYMBOLIC CONSTANT clause

DATA DIVISION

- Literal in a VALUE clause. However, a data item specified in a VALUE clause must be an internal floating-point data item.
- Data-name in a REDEFINES or a RENAMES clause

PROCEDURE DIVISION

- Identifier or literal in a conditional expression. The identifier or literal can be written in a relation condition only.
- Identifiers or literal in an ADD statement
- Identifiers or literal in a COMPUTE statement
- Identifier or literal in a DISPLAY statement

  If a floating-point literal or external floating-point data item is displayed to the user, it is displayed without being converted.

An internal floating-point data item is converted to an external floating-point data item having the format shown below before it is displayed:

  - Single-precision: -.9(8)E-99
  - Long precision: -.9(17)E-99

- Identifiers or a literal in a DIVIDE statement excluding the REMAINDER phrase
- Identifiers or literal in an INITIALIZE statement

  When a floating-point data item is initialized with the INITIALIZE statement, it is initialized as follows:

  - If the INITIALIZE statement does not use the REPLACING phrase, it is initialized to zero.
  - If the REPLACING phrase also includes the NUMERIC phrase, floating-point data items are also initialized.
- Identifiers or literal in a MOVE statement

  When transcribed, the floating-point data item is treated in the same way as the numeric data item of a non-integer.

The precision of an internal floating point differs from the precision of an external floating point and fixed point. When transcribing data between two types, the ranges must be common.

- Identifiers or literal in a MULTIPLY statement
- Identifiers or literal in a SUBTRACT statement
- Data-name in the USING phrase of PROCEDURE DIVISION
- Data-name in the USING phrase of an ENTRY statement
- Identifiers or literal in the USING phrase of a CALL statement. The identifiers or literals cannot be specified in the USING BY VALUE phrase.
- Identifiers or literal in the REPLACING phrase of a COPY statement
- Argument of a numeric function if the argument type is numeric.

Floating-point numbers are implemented through architecture-dependent functions, and their internal formats vary depending on the system. Since values of floating-point numbers are represented internally by binary values, not all values can be correctly represented and some values are represented by approximate values. Consequently, floating-point numbers contain errors in most cases. When using a floating-point number, consider the above points.

2.2 Input-Output Facility

There are three types of input-output facilities:

- Sequential I-O module
- Relative I-O module
- Indexed I-O module

The sequential I-O module processes records in a file according to a set sequence. A file processed by the sequential I-O module is called a "sequential file." Each record in a sequential file is identified by its position at the time the record was written.

The relative I-O module processes any selected record in a file or processes the records in a file according to a set sequence. A file processed by the relative I-O module is called a "relative file." Each record in a relative file is identified by its relative record number.

The indexed I-O module processes any selected record in a file or processes the records in a file according to a set sequence. A file processed by the indexed I-O module is called an "indexed file." Each record in an indexed file is identified by the value of one or more keys.

Include the following entries as required in a program using the input-output facility.

Program Description	Main Facility
ENVIRONMENT DIVISION	
INPUT-OUTPUT SECTION	
FILE-CONTROL paragraph	
File control entry	Associates the file-name with an external medium. Specifies the organization, access mode, relative key, prime record key, alternate record key, and lock mode.
I-O CONTROL paragraph	Specifies the storage area to be shared by several files.
DATA DIVISION	
FILE SECTION	
File description entry	Defines the physical structure of a file.
Record description entry	Defines the structure of a record.
PROCEDURE DIVISION	
Declarative	
USE statement and USE AFTER STANDARD EXCEPTION	Defines the input-output error procedure.
Procedure	
OPEN	Opens a file.
READ	Reads a record from a file.
WRITE	Writes a record to a file.
REWRITE	Rewrites a record in a file.
DELETE (*1)	Deletes a record in a file.
START (*1)	Positions a record pointer in a file.
UNLOCK	Releases a lock on a record.
CLOSE	Closes a file.

*1 : The DELETE and START statements are not included in the sequential file facility.

2.2.1 File Organization

The logical structure of a file on an external medium is called its "organization." There are four file organizations as listed below. The ORGANIZATION clause of the FILE-CONTROL entry specifies which type of organization is to be used.

- Sequential organization
- Line sequential organization
- Relative organization
- Indexed organization

Sequential and line sequence file organizations are incorporated in the sequential file facility. Relative file organizations are handled with the relative file facility. An indexed file is handled by the indexed file facility.

Sequential File Organization

A sequential file consists of records identified according to the positions immediately before and immediately after each record. The logical position of a record is determined when the record is written and does not change except when a record is added at the end of the file.

To use a sequentially organized file, specify the SEQUENTIAL phrase in the ORGANIZATION clause or omit the ORGANIZATION clause.

There are two types of sequentially organized files: record sequential file and print file. The record sequential file type supports input-output handling to and from an external storage device. The print file type outputs data to a printing device. The ASSIGN clause, WRITE statement, and entries in the record description entry determine whether a record sequential file or print file should be used. Refer to the "NetCOBOL User's Guide" for more information on the rules for determining which file type to used.

F

The print file type can edit and output records according to a form descriptor. When using a form descriptor, the FORMAT clause of the FILE-CONTROL entry specifies identifies the data item which contains the form descriptor name.

A print file specifying a FORMAT clause is called a "print file with FORMAT clause." A print file not specifying a FORMAT clause is called a "print file without FORMAT clause."

F

Line Sequential Organization File

A line sequential organized file consists of records delimited by delimiters. One record is counted as one line, and one line consists of printable characters and record delimiters. The logical position of a record is determined when the record is written and does not change except when a record is added at the end of the file.

To use a line sequential file, specify the LINE SEQUENTIAL phrase in the ORGANIZATION clause.

Relative File Organization

A file with relative organization consists of records identified by a relative record number. A relative record number is an integer of 1 or higher. Each record in a relative file is stored in the position indicated by its relative record number when the record is written. For example, a record written with a relative record number of 10 is stored in the area whose relative record number is 10 even if no records are stored in the positions indicated by relative record numbers 1 to 9.

To use relative file organization, specify the RELATIVE phrase in the ORGANIZATION clause. To process records randomly, the relative key item must be specified in the RELATIVE KEY phrase of the ACCESS MODE clause.

Indexed File Organization

An indexed file consists of records and indexes for identifying records. A record in an indexed file is identified by the value of one or more keys defined in the record.

To use an indexed file, specify the INDEXED phrase in the ORGANIZATION clause. Specify the key in the RECORD KEY clause or the ALTERNATE RECORD KEY clause. Specify the prime record key in the RECORD KEY clause and the alternate record key in the ALTERNATE RECORD KEY clause. The prime record key must be specified.

The prime record key and alternate record key are generically called "record keys." The prime record key may or may not be unique within a file. To allow for duplicate primary keys, use the DUPLICATES phrase in the RECORD KEY clause(*).

Alternate record keys may also be unique or may be duplicated within the file. To allow for duplicate alternate record keys, specify the DUPLICATES phrase in the ALTERNATE RECORD KEY clause.

The record key used to access an indexed file is called the "key of reference." The key of reference changes upon execution of an input-output statement.

F

* : Extended functions or functions specific to NetCOBOL.

- RECORD KEY clause

2.2.2 File Connector

A file connector associates a file on an external medium with a program. It is a storage area that holds information about files. It can be used for the following purposes:

- To link a file-name with a file on an external medium.
- To link a file-name and the record area associated with a file.

Use a SELECT statement and the ASSIGN clause in a file control entry to associate a file connector with a file on an external medium. The file connector is established when the OPEN statement is executed.

2.2.3 Operation of Input-Output Statements

File processing is accomplished by combining input-output statements (the OPEN, READ, WRITE, REWRITE, START, DELETE, and CLOSE statements).

Before file processing can begin, the OPEN statement must be executed to open the file. After processing is completed, the CLOSE statement closes the file. The access mode and open mode determine which input-output statements can be executed between the OPEN and CLOSE statements.

Access Mode

The sequence in which records in a file are processed is called the "access mode." There are three access modes: sequential access mode, random access mode, and dynamic access mode. In sequential access mode, reading and writing of records is done in a fixed sequence. In random access mode, reading and writing records is done in a random sequence. In dynamic access mode, processing switches between the sequential and random access modes as appropriate while input-output statements are being executed.

Specify the access mode in the ACCESS MODE clause of the file control entry. Sequentially organized files must be accessed sequentially (i.e., the ACCESS MODE must be SEQUENTIAL).

Open Mode

The mode becoming active when a file is opened is called "open mode." There are four open modes: input mode, output mode, input-output mode, and extension mode. Specify the open mode when a file is opened with the OPEN statement.

Relationship between Open Mode and Input-Output Statement for Sequential Files

The following table shows the relationship between the open mode and input-output statement for sequential files.

Input-Output Statement	Open Mode			
	Input Mode	Output Mode	I-O Mode	Extension Mode
READ	Possible	-	Possible	-
WRITE	-	Possible	-	Possible
REWRITE	-	-	Possible	-

Relationship between Open Mode and Input-Output Statement for Relative Files and Indexed Files

The table below shows the relationship between the open mode and input-output statement for relative files and indexed files.

Access Mode	Input-Output Statement	Open Mode			
		Input Mode	Output Mode	I-O Mode	Extension Mode
Sequential	READ	Possible	-	Possible	-
	WRITE	-	Possible	-	Possible
	REWRITE	-	-	Possible	-
	START	Possible	-	Possible	-
	DELETE	-	-	Possible	-
Random	READ	Possible	-	Possible	-

Access Mode	Input-Output Statement	Open Mode			
		Input Mode	Output Mode	I-O Mode	Extension Mode
	WRITE	-	Possible	Possible	-
	REWRITE	-	-	Possible	-
	START	-	-	-	-
	DELETE	-	-	Possible	-
Dynamic	READ	Possible	-	Possible	-
	WRITE	-	Possible	Possible	-
	REWRITE	-	-	Possible	-
	START	Possible	-	Possible	-
	DELETE	-	-	Possible	-

2.2.4 File Position (Record Pointer) Indicator

A file position or record position indicator is provided for in the concept of COBOL for determining the next record to be processed in a series of input-output operations. The file position indicator has meaning only in a file opened in input mode or input-output mode.

The status of the file position indicator for a sequential file is influenced by execution of a CLOSE statement, OPEN statement, or READ statement.

The status of the file position indicator for a relative or indexed file is influenced by execution of a CLOSE statement, OPEN statement, READ statement, or START statement.

2.2.5 Volume Indicator

A volume indicator is provided in the concept of COBOL for making the current physical volume unique. The volume indicator exists only for a sequential file. The status of the volume indicator is influenced by execution of a CLOSE statement, OPEN statement, READ statement, or WRITE statement.

F

2.2.6 Sharing and Exclusion of Files

COBOL programs can be written so that a physical data file can share its records with one or more COBOL run units (shared mode), or, can be written such that it retains exclusive control of the data file and its records (exclusive mode).

The LOCK MODE clause and OPEN statement determine whether a file is opened in the shared or exclusive mode. Files which specify shared mode in the LOCK MODE clause can be opened either in shared mode or exclusive mode. Files for which the exclusive mode has been specified in the LOCK MODE clause can be opened only in exclusive mode.

Executing the OPEN statement determines whether a physical file can be used in shared mode or exclusive mode.

A file opened in shared mode has a separate file indicator associated with it each time the file is opened in shared mode. This means a single physical file can have two or more file indicators associated with it.

A file opened in exclusive mode has only one file indicator associated with it. This file is then locked and cannot be opened by any other run unit. Executing a CLOSE statement for a file opened in exclusive mode releases the lock on the file and the file can then be opened by another run unit.

The table below indicates the mode (shared or exclusive) of physical files using various specifications.

Specification in LOCK MODE clause	OPEN statement							
	WITH LOCK specified				WITH LOCK not specified			
	INPUT	I-O	OUTPUT	EXTEND	INPUT	I-O	OUTPUT	EXTEND
No specification	Exclusive mode				Shared mode	Exclusive mode		

Specification in LOCK MODE clause	OPEN statement							
	WITH LOCK specified				WITH LOCK not specified			
	INPUT	I-O	OUTPUT	EXTEND	INPUT	I-O	OUTPUT	EXTEND
AUTOMATIC					Shared mode		Exclusive mode	Shared mode
MANUAL(*)					Shared mode		Exclusive mode	Shared mode
EXCLUSIVE					Exclusive mode			

* : MANUAL cannot be specified for a sequential file.

F

2.2.7 Locking a Record

A record read by a READ statement and located in a file opened in shared mode or input-output mode can be locked. Locking a record prevents input-output statements of other file connectors from referencing the record.

If the record is in a sequential file, only one record in one physical file can be locked at one time. If the record is in a relative file or indexed file, any number of records in a single physical file can be locked. To lock only one record, specify AUTOMATIC in the LOCK MODE clause. To lock several records, specify MANUAL in the LOCK MODE clause.

Locking a Record When LOCK MODE IS AUTOMATIC

The file indicator of a file specifying AUTOMATIC in the LOCK MODE clause and opened in input-output mode can maintain a lock on a single record. At execution of a READ statement where WITH LOCK has been specified or where the WITH [NO] LOCK has not been specified, the record read is locked and the lock information in the file indicator is updated. A READ statement specifying WITH NO LOCK does not lock the record.

Executing a READ statement specifying the WITH LOCK phrase prevents any other file indicator from executing a READ, WRITE, REWRITE, or DELETE statement on the locked record. However, it does not prevent the file indicator of a file opened in extension mode from executing a WRITE statement.

The lock on the record is released when any of the following statements is executed for the same file indicator executing the original READ statement:

1. A READ, WRITE, REWRITE DELETE or START statement
2. An UNLOCK statement
3. A CLOSE statement executed either explicitly or implicitly

Locking a Record When LOCK MODE IS MANUAL

The file indicator of a file specifying MANUAL in the LOCK MODE clause and opened in input-output mode can maintain the lock on several records simultaneously. At each execution of a READ statement specifying WITH LOCK, the record read is locked and relevant lock information is added to the file indicator. A READ statement specifying WITH NO LOCK does not lock a record.

Executing a READ statement specifying WITH LOCK prevents any other file indicator from executing a READ, WRITE, REWRITE, or DELETE statement on the set of locked records. However, it does not prevent execution of an input-output statement on a record not locked.

The lock on the records is released when any of the following statements are executed for the same file indicator executing the original READ statement:

1. An UNLOCK statement
2. A CLOSE statement executed either explicitly or implicitly. The lock on several records cannot be released for individual records.

2.2.8 I-O Status

The I-O status is a double byte character area provided for indicating the result of an input-output statement. It is set during execution of a CLOSE, DELETE, OPEN, READ, REWRITE, START, or WRITE statement.

F

More detailed I-O information is available for print files specifying the FORMAT clause.

The I-O status is set each time an input or output statement is executed before execution of any unconditional statement specified in the input-output statement and before execution of a USE AFTER STANDARD EXCEPTION procedure relating to the file.

The I-O status and more detailed information can be referenced by specifying the FILE STATUS clause in the FILE-CONTROL entry.

The first digit position of the I-O status indicates the classification of the I-O status. The table below shows the classifications.

I-O status	Classification	Meaning
0x	Succeed	The input-output statement has been executed successfully.
1x	At end condition	Execution of the sequential access READ statement failed due to the at end condition.
2x (*)	Invalid key condition	Execution of the input-output statement failed due to an invalid key condition.
3x	Permanent error condition	Execution of the input-output statement failed due to an error which prevented continuation of file processing.
4x	Logical error condition	Execution of the input-output statement failed due to a sequence error in execution or because the user went beyond the set limit.
9x	Other error	Execution of the input-output statement failed due to an error not mentioned above.

* : An invalid key condition does not occur in a sequential file.

The I-O status determines whether the USE AFTER STANDARD EXCEPTION procedure associated with the file is executed. If the USE AFTER STANDARD EXCEPTION procedure associated with the file exists and the I-O status is other than "Succeed", the procedure is executed.

At End Condition

An at end condition may occur at execution of a sequential access READ statement. When this condition occurs, execution of the READ statement fails. The file is not affected.

Use the AT END phrase in the READ statement to enable the program to detect whether an at end condition has occurred.

Invalid Key Condition

An invalid key condition may occur at execution of a DELETE, random access READ, REWRITE, START, or WRITE statement for a relative or indexed file. When this condition occurs, execution of the input-statement fails. The file is not affected.

Use the INVALID KEY phrase in the input-output statement to enable the program to detect whether an invalid key condition has occurred.

File Attribute Conflict Condition

A file attribute conflict condition may occur at execution of an OPEN statement. When this condition occurs, execution of the OPEN statement fails. The file is not affected.

2.2.9 Record Format

There are two types of records in a file: fixed-length records and variable-length records. The description of the file and records as defined in the program determines the record format, regardless of the physical layout of the file on it is storage medium.

All fixed-length records in a file have the same number of character positions. Variable-length records in a file can have a different number of character positions.

The RECORD clause and record description entry determines the record format. It also determines the record length written by the WRITE statement and the record length read by the READ statement.

The following table shows the record format, maximum record length, and minimum record length for various descriptions.

Description in RECORD clause	Record format	Minimum record length	Maximum record length
Format 1 RECORD integer-1	Fixed-length	Value of integer-1	Value of integer-1
Format 2 RECORD VARYING [FROM integer-2 [TO integer-3]] [DEPENDING ON data-name-1]	Variable-length	Minimum value of length of record description entry	Maximum value of length of record description entry or integer-3, whichever is greater
Format 3 RECORD integer-4 TO integer-5 or RECORD clause is not specified	Variable-length in the following cases: In all other cases, the record format is fixed-length.	Minimum value of length of record description entry. (*1)	Maximum value of length of record description entry. (*1)
	- Two or more record description entries of different lengths have been written.		
	- An OCCURS DEPENDING ON clause has been written in the record description entry.		
	- A CHARACTER TYPE clause has been written in the record description entry. (*2)		

*1: The minimum value and maximum value of the length of the record description entry are the same for a fixed-length record.

*2: It is native to [HP], [UXP/DS], [Solaris], [Linux], [LinuxIPF] and [Linux64].

2.2.10 Record Area

The storage area allocated for a record description entry in the FILE SECTION is called a "record area." Several record description entries can be associated with a single file but the record area is allocated to the file, not to the record description entries. The record area size is the same as the maximum record length of the record associated with the file.

A record area can be shared by several files in a program. To enable the record area to be shared, specify the files to share the record area with the SAME RECORD AREA clause of the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION.

2.2.11 LINAGE-COUNTER Special Register

Writing a LINAGE clause in the file description entry of a sequential file creates the LINAGE-COUNTER special register. LINAGE-COUNTER is treated as an integer item the same size as integer-1 or data-name-1 in the LINAGE clause and does not have symbols.

LINAGE-COUNTER counts the number of lines held and printed by the I-O-control system. Only statements in the PROCEDURE DIVISION can reference the value of LINAGE-COUNTER. The user cannot set the value of LINAGE-COUNTER.

2.3 Inter-program Communication Module

The inter-program communication module allows communication among various programs. This communication may take four forms: transfer of control, the passing of parameters, the reference to common data and the reference to common files.

Executing a CALL statement calls a program in another separately compiled program or another program in the same separately compiled program. The program executing the CALL statement is called the "calling program." The program to which control is passed by execution of the CALL statement is called the "called program."

The inter-program communication module not only passes control back and forth between the calling program and the called program but also passes parameters and shares data and or files.

Use the following descriptions as required in a program which uses the inter-program communication module.

Program Description	Main Facility
IDENTIFICATION DIVISION	Specifies the name of the program and gives the following attributes for the program - The initial attribute - The common attribute - The recursive attribute ([Winx64], [Linux64] and [.NET] only)
DATA DIVISION	
FILE SECTION	
File description entry	Sets the external attribute in a file connector. Sets the global attribute in the file-name and data-name
Record description entry	Sets the external attribute in a data record. Sets the global attribute in a data-name
LINKAGE SECTION	
77-level description entry and record description entry	Defines the parameters to be accepted by a called program
WORKING STORAGE SECTION	
Record description entry	Sets the external attribute in a data record. Sets the global attribute in a data-name
BASED-STORAGE SECTION and CONSTANT SECTION	
Record description entry	Sets a global name in a data-name
LOCAL-STORAGE SECTION	
77-level description entry and record description entry	[Winx64][Linux64][.NET] Defines the automatic item
Procedure name	
HEADER OF PROCEDURE DIVISION	Specifies the parameters to be accepted by a called program

Program Description			Main Facility
	Procedure portion		
		CALL	Calls another program
		CANCEL	Initializes another program
		ENTRY	Specifies the secondary entry name. Specifies the parameters to be accepted
		EXIT PROGRAM	Resumes the calling program
End program header			Specifies the end of a program

2.3.1 Accessing and Returning to a Program

Execute a CALL statement to transfer control to another program. To return to the calling program from the called program, execute the EXIT PROGRAM statement.

Run Unit

A group of object programs mutually linked at execution is called a "run unit." A run unit consists of a single separately compiled program or a group of several separately compiled programs.

Main Program and Sub-program

The COBOL program first activated in a separately compiled program is called the "main program." Another COBOL program called from the main COBOL program is called a "sub-program." A nested program is a sub-program regardless of whether the outermost program in the separately compiled program is the main program or a sub-program.

Executing a Called Program

Execution of a CALL statement makes the called program ready for execution. Execution of the called program begins at one of the following statements depending on the description in the CALL statement.

1. When a program-name or program-name literal has been specified in the CALL statement, execution starts at the first statement in the procedure portion of the PROCEDURE DIVISION.

F

2. When a secondary entry point name has been specified in the CALL statement, execution starts at the first statement following the ENTRY statement. "Secondary entry point name" refers to the literal written in the ENTRY statement. An ENTRY statement can only be specified in the outermost program of a separately compiled program.

Executing and Terminating a Called Program

To terminate execution of a called program, execute an EXIT PROGRAM or STOP RUN statement. These statements are generically called "program end statements."

To end a called program and return to the calling program, execute an EXIT PROGRAM statement. To end a program and return to the operating system, execute a STOP RUN statement.

2.3.2 Global Name and Local Name

A file-name (excluding the file-name of a sort-merge file), record-name, data-name (excluding the data-name of a screen item), and condition-name may have either a global attribute or a local attribute.

A name having the global attribute is called a "global name" and a name having the local attribute is called a "local name." A global name can be referenced in a program with a defined name and all programs directly or indirectly contained in the defined program. A local name can only be referenced in a program with a defined name.

Specifying the GLOBAL clause in a file description entry makes the following names global names:

- The file-name written in the file description entry
- All data-names, record-names, and condition-names in a record description entry associated with the file description entry

Specifying the GLOBAL clause in a data description entry of level-number 01 makes the following names global names:

- A data-name (record-name) written in the data description entry whose level-number is 01
- All data-names and condition-names subordinate to the data description entry whose level-number is 01

File-names, record-names, data-names, and condition-names not defined as global names become local names.

2.3.3 External Attribute and Internal Attribute

A data item or file connector (excluding the file connector of a sort-merge file) may have an external attribute or an internal attribute. Set the external attribute to enable several programs in a run unit to share an area relating to a data item or file connector.

Specifying the EXTERNAL clause in a file description entry sets the external attribute in the file connector corresponding to the file description entry. A file connector having the external attribute is called an "external file connector."

Writing an EXTERNAL clause in the data description entry of level-number 01 sets the external attribute in the data item defined in the data description entry and all subordinate data items. A record configured in a data item having the external attribute is called an "external data record."

One storage area is allocated to all external file connectors having the same file-name in a run unit. One storage area is also allocated to all external data records having the same record-name in a run unit. External file connectors and external data records can be referenced from any program in the run unit. When referencing an external file connector or external data record in a nested program, the name referenced must be defined as a global name.

Data items and file connectors in which the external attribute has not been set are given the internal attribute. Data items and file connectors having the internal attribute can be referenced only in one separately compiled program. A file connector which does not have the external attribute is called an "internal file connector."

The external attribute cannot be set in the following items. These items are always given the internal attribute.

F - A data item defined in BASED-STORAGE SECTION

- A data item defined in CONSTANT SECTION
- A data item defined in LINKAGE SECTION
- A data item defined in REPORT SECTION
- A data item defined in LOCAL-STORAGE SECTION ([Winx64], [Linux64] and [.NET] only)
- A data item defined in the record description entry of a sort-merge file

F - A screen item

F The compiler does not set the external attribute in an index-name even if an EXTERNAL clause has been written in a record description entry including a data description entry where the INDEXED BY phrase has been written.

F 2.3.4 External Name and Internal Name

An external program has both an external name and internal name. A nested program has only an internal name.

An external name, which is the name given to an external program, is used when referencing or being referenced by other programs. The external name can be defined and referenced by a literal. An external name is a way to use names that are outside the scope of the COBOL character set when linking with other languages.

The internal name must be defined according to the rules of user-defined words. An internal name is used to express an arbitrary program-name within a compilation unit.

When an external name is not explicitly defined for an external program, an external name having the same name as the internal name is implicitly defined. In this document, any name that is not referred to as an external name is an internal name.

External names are native to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

2.3.5 Program Activation

A program can be activated from the OS or via a call from another program. A program may be activated recursively, and it stays in the active state until either of the following is executed:

- STOP statement
- An implicit or explicit EXIT PROGRAM statement within the same called program.

When the program is activated, the PERFORM statement controls default to their initial state.

2.3.6 The Common Attribute

The common attribute can be set in a nested program. A nested program having the common attribute can not only be called from other programs which directly contain the nested program, but also from a program directly or indirectly contained in the program that contains the nested program. A nested program without the common attribute can be called only from a program that directly or indirectly contains the nested program.

To set the common attribute in a nested program, specify the COMMON phrase in the PROGRAM-ID paragraph of the IDENTIFICATION DIVISION. A nested program having the common attribute is called a "common program."

2.3.7 Initial State of a Program

When a program is first loaded into memory, some initialization will occur. The state of the program at this stage is called the "initial state of program." When a program is first called in a run unit, the program is initialized to the following states.

1. A data item defined in the following is set to its initial state:

   - WORKING-STORAGE SECTION
   - LOCAL-STORAGE SECTION ([Winx64], [Linux64] and [.NET] only)

   A data item that specifies a VALUE clause and any data item subordinate to such a data item is initialized to the value in the VALUE clause. The initial value for a data item outside the scope of the VALUE clause is not defined.

2. An internal file connector is set to a state other than to the open state.
3. A GO TO statement referenced in the ALTER statement is set to the initial state.

To set the program in its initial state the second and subsequent times it is called in the run unit, use one of the following methods:

- Set the initial attribute in the program.
- Execute a CANCEL statement.

Initial Attribute of a Program

An initial attribute can be set in a program. A program having the initial attribute is always initialized to its initial state each time it is called.

To set the initial attribute in a program, specify the INITIAL clause in the PROGRAM-NAME paragraph of the IDENTIFICATION DIVISION. Specifying the INITIAL clause sets the initial attribute in a program and in all programs contained directly or indirectly therein. Both the initial and common attributes can be set in a nested program.

A program having the initial attribute is called an "initial program." An initial program is always set to its initial state whenever it or any secondary entry points(*) are called by a CALL statement.

F

* : Extended functions or functions specific to NetCOBOL.

- secondary entry points

Initialization by CANCEL Statement

A CANCEL statement will force re-initialization of a program when it is next called. Executing the CANCEL statement sets the program specified in the CANCEL statement and any programs contained directly or indirectly therein to the initial state the next time it is called.

F

2.3.8 Recursive Attribute of the Program

The recursive attribute can be assigned to a program by specifying the RECURSIVE clause in the IDENTIFICATION DIVISION.

A recursive program can invoke itself directly or indirectly, and it initializes data in the LOCAL-STORAGE SECTION each time the program is invoked.

The recursive attribute is available for [Winx64], [Linux64] and [.NET].

2.3.9 Passing Parameters to the Called Program

To pass a parameter between a calling program and a called program, specify the parameter as follows:

1. Specify the parameter to be passed in the USING phrase of the CALL statement in the calling program.

F

   If the called program returns a value to the calling program, specify a data item name to receive the returned data in the RETURNING phrase of the CALL statement. When calling a C program, a function value can be returned in the RETURNING phrase.

2. In the called program, specify the parameter to be passed in the PROCEDURE DIVISION header or in the USING phrase of the ENTRY statement(*).

F

   In addition, the RETURNING phrase can be used to return a value to the calling program.

   Data items for USING phrase and RETURNING phrase(*) are defined in the Linkage Section. Parameters are associated between the calling program and the called program by the sequence they were written in the USING phrase. The names of corresponding parameters need not be the same.

F

* : Extended functions or functions specific to NetCOBOL.

- ENTRY statement
- RETURNING phrase

Specify whether the value of a parameter changed in a called program is to be returned in the USING phrase of the CALL statement as follows:

1. If the called program is to update the parameter that was passed to it, the calling program's CALL statement must specify the parameter to update in the USING BY REFERENCE phrase. Or, you may omit BY REFERENCE of the USING phrase.
2. When the value of a parameter is not to be changed by the called program, specify the parameter in
   - USING BY CONTENT phrase or the USING BY VALUE phrase(*).

F

   Some languages, such as C, expect parameters to be passed by value, and in these cases specify the parameters in the USING BY VALUE phrase.

F

* : Extended functions or functions specific to NetCOBOL.

- ENTRY statement
- RETURNING phrase
- USING BY VALUE phrase

2.3.10 Scope of Names

Within a main program and each of its contained subprograms, data and files are normally private, which means that they are accessible only to the program in which they are contained. However, COBOL has provisions for sharing data and files among programs.

Referencing User-defined Words in the Program That Defined Them

The following user-defined words are private to the program that defines them, and cannot be referenced by other programs. Other programs may define user-defined words with the same name, but these words are private to the programs in which they are defined.

- Paragraph-name
- Section-name
- Report-name
- File-name of a sort-merge file

F
- Data-name of a screen item

User-defined Words of All Separately Compiled Programs

The following user-defined words can be referenced in all separately compiled programs. However, the name to be referenced must be associated with the separately compiled program at compilation.

- Library-name
- Text-name

User-defined Words and Contained Programs

The following user-defined words can be referenced in the program that defined them and in any programs contained directly or indirectly in the defining program:

- Alphabet-name
- Class-name
- Condition-name (condition-name defined in the ENVIRONMENT DIVISION)
- Mnemonic-name
- Symbolic-character

F
- Symbolic constant
- Positioning unit-name
- Print mode name

User-defined Words and the Global Attribute

For the following user-defined words the scope of the name changes depending on whether the name has the global attribute:

- Condition-name (condition-name defined in DATA DIVISION)
- Data-name(excluding a data-name of a screen item(*))
- Record-name
- File-name (excluding the file-name of a sort-merge file)

F

* : Extended functions or functions specific to NetCOBOL.

- screen item

When these user-defined words have the global attribute, these words can be referenced in the program that defined them and in any programs contained directly or indirectly in the defining program. When the global attribute has not been set, these words can only be referenced in the program that defines them.

Contained programs may define user-defined words with the same name as user-defined words that have the global attribute.

For example, program A directly contains program B. User-defined word "X" is defined in two or more programs. If program B references user-defined word "X", the following rules are applied in the order shown to identify which user-defined word "X" is to be referenced:

1. If user-defined word "X" is defined in program B, the system references word "X" in program B.
2. If user-defined word "X" is not defined in program B, the system searches the programs starting from program A and moving outward until it finds the program where user-defined word "X" has been defined. When it first finds the program where word "X" has been defined, it references the area of word "X" in that program.

Scope of Index-name Name

When an index-name has been associated with a table and the data-name affixed in the table has the global attribute, the index-name is also given the global attribute. The scope of the name of an index-name is the same as the scope of the name of a data-name affixed to a corresponding table. See the above item "User-defined words for the scope of the name changes depending on the global attribute."

2.3.11 Scope of a Program-name and Secondary Entry Point Names(*)

Within a separately compiled program, the program-name and any secondary entry point names(*) cannot be the same.

Within a single run unit composed of two or more separately compiled programs, the program-name of the outermost program and secondary entry point names(*) in each separately compiled program must not be the same. However, a nested program may have the same program-name or secondary entry point name(*) as separately compiled programs.

A program-name is declared by the program-name paragraph in the program header. There are two program-names for every program: an external program name(*) and a nested program name. The nested program name is referenced when calling a program within the same compilation unit as the program issuing the CALL or CANCEL statement. The external program name(*) is referenced when calling a program that is not within the same compilation unit as the program that issues the CALL or CANCEL statement. A nested program name is specified by the END PROGRAM header.

All external program names(*) within a run unit must be unique. Nested program names within a run unit need not be unique. However, duplicate nested program names within a run unit must reside within separate compilation units.

CALL and CANCEL statements may reference program-names and secondary entry point names(*) of different programs. The program-names and secondary entry point names(*) that can be referenced in a CALL or CANCEL statement are as follows:

1. Nested program-names not having the common attribute can be referenced only from a program that contains the nested program directly or indirectly.
2. Nested program-names that have the common attribute can be referenced from the following:
   - A program that contains the nested program directly or indirectly
   - A program contained either directly or indirectly in the program that directly contains the nested program

   A nested program with the common attribute cannot CALL or CANCEL itself nor may a program contained in the nested program CALL or CANCEL the outer nested program-name.
3. External program names(*) and secondary entry point names(*) of separately compiled programs can be referenced from any program in the run unit. However, these names cannot be referenced from programs that are contained directly or indirectly in the program.

F

* : Extended functions or functions specific to NetCOBOL.

- secondary entry point names
- external program names

F

2.3.12 PROGRAM-STATUS and RETURN-CODE Special Registers

The special registers PROGRAM-STATUS and RETURN-CODE are automatically created for programs which do not have the RETURNING phrase in the PROCEDURE DIVISION header. PROGRAM-STATUS and RETURN-CODE are synonymous. PROGRAM-STATUS is used to pass a return code to the operating system or the program that called it. PROGRAM-STATUS is implicitly defined as a numeric data item with a PICTURE of "S9(9)" and a USAGE of "COMPUTATIONAL-5." However, for [Winx64], [LinuxIPF] and [Linux64], PROGRAM-STATUS is implicitly defined as a numeric data item with a PICTURE of "S9(18)" and a USAGE of "COMPUTATIONAL-5."

The value of PROGRAM-STATUS when the program begins execution is zero. After moving a return code to PROGRAM-STATUS, two options are available:

1. If the called program issues a STOP RUN statement, control is returned to the operating system, and the return code is reported to the operating system.
2. The called program may issue an EXIT PROGRAM statement. In this case, control is returned to the calling program, and the return code is stored in PROGRAM-STATUS in the calling program.

2.4 Sort-Merge Module

The Sort-Merge module consists of the Sort and the Merge modules.

The Sort module arranges records in a file according to a specific key. The SORT statement invokes the Sort module and arranges the records in sequence.

The Merge module merges files whose records are prearranged according to a specific key. The MERGE statement invokes the Merge module and merges the files.

A sort-merge file must be defined before the Sort-Merge module can be used. The sort-merge file is created internally by the COBOL runtime system, and it is not associated with an external medium.

Use the following descriptions as required in a program that uses the Sort-Merge module.

Program Description	Main Facility
ENVIRONMENT DIVISION	
INPUT-OUTPUT SECTION	
FILE-CONTROL paragraph	
File control entry	Specifies a sort-merge file.
I-O CONTROL paragraph	Specifies the storage area shared by several files containing a sort-merge file.
DATA DIVISION	
FILE SECTION	
Sort-merge file description entry	Defines the physical structure of the sort-merge file.
Record description entry	Defines the structure of a record in the sort-merge file.
PROCEDURE DIVISION	
Procedure portion	
SORT	Arranges the record in sequence.
MERGE	Merges two or more files.
Input procedure	

Program Description	Main Facility
RELEASE	Delivers a record to the first step in a sort operation.
Output procedure	
RETURN	Accepts a record from the last step in a sort operation. Accepts a record merged by the merge operation.

2.4.1 Sorting Methods

There are four sorting methods:

1. A method that directly sorts the records in a file and directly writes the result to another file. To use this method, write the USING and GIVING phrase in the SORT statement.
2. A method that performs special processing on records before they are sorted, then sorts the records and directly writes the result to another file. To use this method, write the INPUT PROCEDURE phrase and GIVING phrase in the SORT statement.
3. A method that directly sorts records in a file and then performs special processing on the sorted records. To use this method, write the USING phrase and OUTPUT PROCEDURE phrase in the SORT statement.
4. A method that performs special processing on records before they are sorted, then sorts the records and performs special processing on the sorted records. To use this method, write the INPUT PROCEDURE phrase and OUTPUT PROCEDURE phrase in the SORT statement.

2.4.2 Merging Methods

The records in files that are to be merged must be ordered by a merge key before the files can be merged. There are two merging methods:

1. A method that directly merges records in a file and directly writes the result to another file. To use this method, write the USING phrase and GIVING phrase in the MERGE statement.
2. A method that directly merges records in a file and then performs special processing on the merged records. To use this method, write the USING phrase and OUTPUT PROCEDURE phrase in the MERGE statement.

2.4.3 SORT Input Procedure

A SORT statement containing the INPUT PROCEDURE phrase requires that an input procedure be written in the PROCEDURE DIVISION. The input procedure repeats the following processes:

1. Performs special processing on records before they are sorted. For example, it edits and selects the records.
2. Executes the RELEASE statement for these records. Executing the RELEASE statement causes the records to be written to the sort-merge file.

After the input procedure has ended, the records written to the sort-merge file are sorted by a series of RELEASE statements.

2.4.4 Sort-Merge Output Procedures

A SORT or MERGE statement containing the OUTPUT PROCEDURE phrase requires that an output procedure must be written in the PROCEDURE DIVISION. The output procedure is executed after sorting or merging of sort-merge files. The output procedure repeats the following processes:

1. Executes the RETURN statement causes records are read from the sort-merge file.
2. Performs special processing on the records read from the sort-merge file. For example, it edits and selects the records.

2.4.5 Sort-merge File

A sort-merge file is created and allocated internally for sorting and merging operations.

Unlike files used with the file module, the sort-merge file does not use the label, blocking, buffer area, and reel concepts.

A sort-merge file can be referenced only in a MERGE, RELEASE, RETURN, or SORT statement.

F

2.4.6 Special Register

2.4.6.1 SORT-STATUS

The special register SORT-STATUS is automatically created for the sort-merge module. SORT-STATUS is treated as a numeric data item defined as "PICTURE S9(4) COMPUTATIONAL-5." It can be used to reference the return code of a sort or merge operation or to suspend a sort or merge operation.

The SORT-STATUS initial value is zero. The value is automatically initialized when the SORT or MERGE statement is first executed.

SORT-STATUS Values

At the completion of the SORT or MERGE statement, a return code is moved to SORT-STATUS. The value of SORT-STATUS is one of the following values:

- 0 : Indicates the sort or merge operation ended normally.
- 16 : Indicates the sort or merge operation did not end normally.

If the program does not contain a reference to SORT-STATUS and its value is other than zero, an error message is output to the system console and the program ends abnormally.

Suspending a Sort or Merge Operation

When a program contains a SORT statement with the INPUT PROCEDURE or OUTPUT PROCEDURE phrase, or a MERGE statement with the OUTPUT PROCEDURE phrase, a sort or merge operation can be suspended by moving a value to SORT_STATUS.

Moving 16 to SORT-STATUS and executing a RELEASE statement in the input procedure shifts control to the end of the SORT statement.

Moving 16 to SORT-STATUS and executing a RETURN statement in the output procedure shifts control to the end of the SORT or MERGE statement.

2.4.6.2 SORT-CORE-SIZE

The special register SORT-CORE-SIZE is automatically created for the sort-merge module. SORT-CORE-SIZE is treated as a numeric data item defined as "PICTURE S9(8) COMPUTATIONAL-5." You can specify memory size used by PowerBSORT SORT/MERGE processing in KB. Please refer PowerBSORT User's Guide for details on the use of this value.

The SORT-CORE-SIZE special register functions in the same manner as the runtime option "smsize" and compile option "SMSIZE".

If you specify more than one them at the same time, the special register "SORT-CORE-SIZE" has the highest precedence, run-time option "smsize" is second, and compile option "SMSIZE" last.

For example:

"MOVE 32 to SORT-CORE-SIZE" means that the memory size is 32KB(32768 bytes).

Note

SORT-CORE-SIZE is a function specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

2.5 Source Text Manipulation Module

The source text manipulation module fetches and replaces portions of program source code at compilation.

The Source Text Manipulation Module utilizes the COPY and REPLACE statements. These statements are processed before any other statements at compilation time. These statements are only processed at compilation time and are ignored during execution.

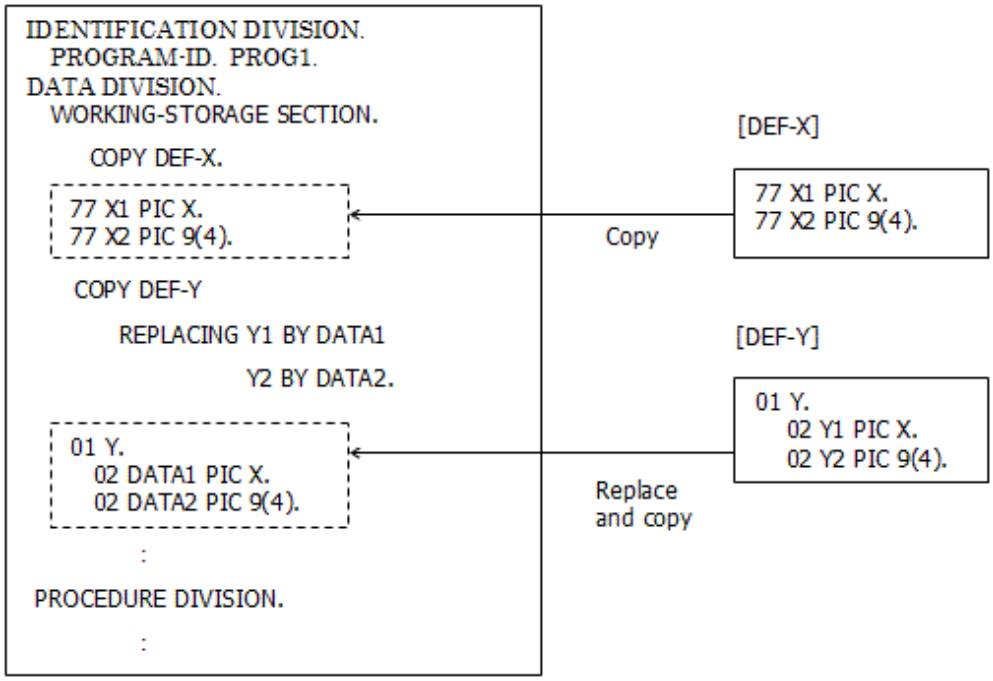
The COPY statement is used to copy source code into the program specifying the COPY statement from a COBOL library. The COBOL library is created separately from the program that specifies the COPY statement and is associated with the program at compilation time.

The REPLACE statement is used to replace portions of program source code.

The COPY and REPLACE statements may be written in any location within a program or COBOL library.

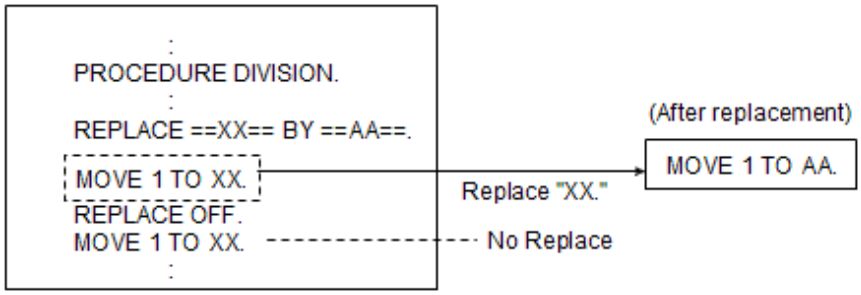
Example of COPY Statement

An example of the COPY statement is given below.



Example of REPLACE Statement

Use the REPLACE statement in pairs: specify the REPLACE statement at the beginning of the area where you want replacement to occur and the REPLACE OFF statement at the end of the area. An example of the REPLACE statement follows.



F

2.6 Presentation File Module

The presentation file module inputs and outputs data as well as sends and receives messages via the presentation service control system or the message control system. It has the following functions:

- Formats and edits data on a computer monitor using a screen descriptor.
- Outputs documents to a printing device using a form descriptor.
- Sends and receives messages between a hardware unit and a program.

Note

For [LinuxIPF], [Linux64] and [.NET], only the "function for outputting documents to a printing device using a form descriptor" can be used.

A screen form descriptor specifies the data layout used by a display or printer. It is created separately from the program that uses it.

The presentation file module uses a presentation file to input and output data and to send and receive messages. It forms the interface between a hardware unit and a program. Executing an input-output statement of a presentation file causes the file to input or output data or to send or receive a message.

Use the following descriptions as required in a program that uses the presentation file module.

Program Description	Main Facility
ENVIRONMENT DIVISION	
INPUT-OUTPUT SECTION	
FILE-CONTROL paragraph	
File control entry	Associates the file-name of the presentation file with an external medium. Specifies the physical attribute for each destination type.
I-O CONTROL paragraph	Specifies processing methods specific to a presentation file, such as handling data items that extend over several conversations.
DATA DIVISION	
FILE SECTION	
File description entry	Defines the physical structure of the presentation file.
Record description entry	Defines the structure of a record in the presentation file.
PROCEDURE DIVISION	
Declarative	
USE statement and USE AFTER STANDARD EXCEPTION	Defines the input-output error procedure.
Procedure portion	
OPEN	Opens a file.
READ	Reads a record from a file.
WRITE	Writes a record to a file.
CLOSE	Closes a file.

2.6.1 Destination Type

Executing an input-output statement to a presentation file causes the file to input and output data and to send and receive messages. The destination type specifies which hardware unit the input-output statement is directed to.

There are the following destination types:

- Display

- Printer
- Other logical units

The destination type is specified in the SYMBOLIC DESTINATION clause of the file control entry. A destination type must be specified before a CLOSE, OPEN, READ, or WRITE statement can be executed.

2.6.2 Screen Form Descriptor

The screen layout of the display and the layout of a document to be printed can be defined separately from the program in a screen form descriptor. The screen form descriptor name is specified in the FORMAT clause of the file control entry.

The screen form descriptor edits records when the program is executed. Executing a WRITE statement causes the records specified in the WRITE statement to be edited in accordance with the screen form descriptor and to be output to a hardware unit. Executing a READ statement causes data input from a hardware unit to be edited in accordance with the screen form descriptor and to be returned to the program.

The data description for data passed between a hardware unit and the program can be copied from the screen form descriptor.

2.6.3 File Organization and Access Mode

This section explains the file organization and access mode of a presentation file.

A presentation file has sequential organization. Specify the ORGANIZATION IS SEQUENTIAL phrase (or omit the ORGANIZATION clause entirely) in the file control entry of the presentation file description.

A presentation file can only be accessed sequentially. Specify the ACCESS MODE SEQUENTIAL phrase (or omit the ACCESS MODE clause entirely) in the presentation file control entry .

2.6.4 Presentation File Input-Output Statements

Processing of a presentation file is accomplished by combining input-output statements (OPEN, READ, WRITE, and CLOSE statements).

An OPEN statement must be executed to open the presentation file before any processing can be performed. After processing, execute a CLOSE statement to close the file. The input-output statements used between the OPEN and CLOSE statements depend on the open mode.

Relationship Between Open Mode and Input-Output Statement for Presentation Files

The following table shows the relationship between the open mode and the input-output statements that can be used for a presentation file.

Input-Output Statement	Open Mode		
	Input Mode	Output Mode	I-O Mode
READ	Possible	-	Possible
WRITE	-	Possible	Possible

2.6.5 I-O Status

The I-O status is a 2-character area used to indicate the result of an input-output statement. It is set during execution of a CLOSE, OPEN, READ, or WRITE statement.

More detailed I-O information is provided in a 4-character area that indicates the result of the input-output statement.

The I-O status is set each time an input or output statement is executed, before execution of any unconditional statements specified in the input-output statement, and before execution of a USE AFTER STANDARD EXCEPTION procedure relating to the file.

The I-O status and more detailed information can be referenced by specifying the FILE STATUS clause in the FILE-CONTROL entry.

The first digit of the I-O status indicates the classification of the I-O status as follows:

I-O status	Classification	Meaning
0x	Succeed	The input-output statement has been executed successfully.
1x	At end condition	Execution of a READ statement failed due to the at end condition.
3x	Permanent error condition	Execution of the input-output statement failed due to an error which prevented continuation of file processing.
4x	Logical error condition	Execution of the input-output statement failed due to a sequence error in execution.
9x	Other error	Execution of the input-output statement failed due to an error other than that mentioned above.

The I-O status determines whether the USE AFTER STANDARD EXCEPTION procedure associated with the file is executed. If the USE AFTER STANDARD EXCEPTION procedure associated with the file exists and the I-O status is other than "Succeed", the procedure is executed.

At End Condition

An at end condition may occur when a READ statement is executed. When this condition occurs, execution of the READ statement fails, and the file is not affected.

Use the AT END phrase in the READ statement to enable the program to detect whether an at end condition has occurred.

2.6.6 Presentation File Special Registers

When creating a screen form descriptor, define not only the screen layout of the document but also the data items to be used by both the program and the hardware unit. If required, also specify an item controller for the data item. An item controller is an area for passing detailed information relating to data item input or output between a hardware unit and the program. A program can reference and set the contents of item controllers by using special registers.

When data items having item controllers have been specified in a record description entry associated with a presentation file, the following seven special registers are created for each data item:

- EDIT-MODE
- EDIT-OPTION
- EDIT-OPTION2
- EDIT-OPTION3
- EDIT-COLOR
- EDIT-STATUS
- EDIT-CURSOR

These seven special registers are treated as single byte alphanumeric data items.

These special registers can only be referenced in the PROCEDURE DIVISION. They are associated with a data item having an item controller, so they must be qualified before they can be referenced. See the "Uniqueness of Reference" section in Chapter 1, "General Rules" for information on qualifying a special register.

The functions of the presentation file special registers are explained below. Refer to "NetCOBOL User's Guide" for information on item controllers and the values used in each special register.

1. The EDIT-MODE special register specifies the processing mode of the data item when a WRITE statement is executed. It can specify whether a data item in a record is to be processed and whether a National data item is to be displayed in the National character set or as an alphanumeric character.
2. The EDIT-OPTION special register controls the appearance of the data item when a WRITE statement is executed. Possible options are blinking, highlighting, reverse video, and underlining of the data item in the WRITE statement.

3. The EDIT-OPTION2 special register specifies processing option 2 (background color) for the data item of identifier-1 when records are written in a presentation file by the WRITE statement. EDIT-OPTION2 is specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].
4. The EDIT-OPTION3 special register specifies processing option 3 (shadow) for the data item of identifier-1 when records are written in a presentation file by the WRITE statement. EDIT-OPTION3 is specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET]
5. The EDIT-COLOR special register controls the data item's color and brightness when it is output.
6. The EDIT-STATUS special register controls the input mode of the data item at execution of a READ statement. It can specify whether a data item is to be processed. If the READ statement is executed normally, the I-O status of the data item (indicating normal input or abnormal input) is moved to EDIT-STATUS.
7. At execution of a READ statement, the EDIT-CURSOR special register determines positioning of the cursor for the data item to the system.

2.7 Intrinsic Function Module

The intrinsic function module performs specific processing based on the values of several arguments and returns the result to the program. The value returned by the function is called the "function value."

The functions listed below are used in statements in the PROCEDURE DIVISION.

The following table lists the intrinsic functions.

Function name	Function value
ACOS	Reverse cosine of argument
ACP-OF	Argument converted to ANSI code page (ACP) character string. ([.NET])
ADDR	Leading address of argument
ANNUITY	Equal payment amount for each quarter
ASIN	Reverse sine of argument
ATAN	Reverse tangent of argument
CAST-ALPHANUMERIC	National data item or National edited data item converted to Alphanumeric data item
CHAR	A single character in the order of the argument according to the collating sequence
COS	Cosine of argument
CURRENT-DATE	Current date and time
DATE-OF-INTEGER	Integer format date converted to standard format date
DAY-OF-INTEGER	Integer format date converted to annual format date
DISPLAY-OF	National characters replaced with the corresponding alphanumeric characters. ([Win32], [Winx64], [Solaris], [Linux64], [.NET])
ENUM-AND	Applies the Boolean operator AND to the list of arguments. ([.NET])
ENUM-NOT	Applies the Boolean operator NOT to the list of arguments. ([.NET])
ENUM-OR	Applies the Boolean operator OR to the list of arguments. ([.NET])
FACTORIAL	Factorial of argument
INTEGER	Maximum integer not exceeding value of argument
INTEGER-OF-DATE	Standard format date converted to integer format date
INTEGER-OF-DAY	Annual format date converted to integer format date
INTEGER-PART	Integer part of argument
LENG	Size of argument (number of bytes)

Function name	Function value
LENGTH	Length of argument (number of character positions or national character positions)
LOG	Natural logarithm of argument
LOG10	Common logarithm of argument
LOWER-CASE	Uppercase alphabetic characters in argument replaced by lowercase alphabetic characters
MAX	Maximum value in list of argument
MEAN	Arithmetic mean of list of argument
MEDIAN	Central value of list of argument
MIDRANGE	Arithmetic mean of minimum value and maximum value in list of argument
MIN	Minimum value in list of argument
MOD	Integer value of argument 1 modeled on argument 2
NATIONAL	COBOL character set converted to National character set.
NATIONAL-OF	Alphanumeric characters replaced with the corresponding National characters. ([Win32], [Winx64], [Solaris], [Linux64], [.NET])
NUMVAL	Character-string in numeric literal format converted to a numeral
NUMVAL-C	Character-string in numeric literal format (including a currency sign or comma) converted to a numeral
ORD	The order of an argument according to the collating sequence
ORD-MAX	Position of argument having the maximum value
ORD-MIN	Position of argument having the minimum value
PRESENT-VALUE	Current value of each end of quarter
RANDOM	Pseudo-random number
RANGE	Difference between maximum value and minimum value in list of argument
REM	Remainder after dividing argument 1 by argument 2
REVERSE	Character-string of argument in reverse order
SIN	Sine of argument
SQRT	Square root of argument
STANDARD-DEVIATION	Standard deviation of list of argument
STORED-CHAR-LENGTH	Number of character positions of valid characters ([Win32], [Winx64],[Solaris], [Linux],[LinuxIPF],[Linux64] and [.NET])
SUM	Sum of list of argument
TAN	Tangent of argument
UCS2-OF	Character encode mode changed to UCS2 ([Win32],[Winx64], [Solaris],[Linux], [LinuxIPF],[Linux64] and [.NET])
UNICODE-OF	The argument value is replaced by a UNICODE character string. ([.NET])
UPPER-CASE	Lowercase alphabetic characters in argument replaced by uppercase alphabetic characters
UTF8-OF	Character encode mode changed to UTF8 ([Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET])
VARIANCE	Variance of list of argument
WHEN-COMPILED	Program compile date and time

F

2.8 Screen Handling Module

The screen handling module displays data on a display and accepts input data from the user via the screen control system. The SCREEN SECTION in the DATA DIVISION defines the screen position where the data is to be displayed and the screen position where the data is to be input. The SCREEN SECTION not only defines the arrangement of data on the screen but also defines the display color and input-output attributes.

Note

The screen handling module cannot be used in [Linux], [LinuxIPF], [Linux64] or [.NET].

Use the following descriptions as required in a program using the screen handling module.

Program Description	Main Facility
ENVIRONMENT DIVISION	
CONFIGURATION SECTION	
SPECIAL-NAMES paragraph	Specifies the data-name for accepting the cursor position and screen input status.
DATA DIVISION	
SCREEN SECTION	
Screen description entry	Defines the attribute of areas on the screen.
PROCEDURE DIVISION	
Procedure portion	
DISPLAY	Displays the screen.
ACCEPT	Inputs data from the screen.

2.8.1 Screen and Screen Item

The screen can be regarded as an area of a fixed size for displaying combinations of lines and columns. Several logical screens can be combined on one physical screen.

2.8.1.1 Screen Data Description Entry

A screen is defined by specifying one or more screen items in a SCREEN SECTION data description entry. The location of data on the screen is determined by specifying line and column numbers in the screen item. The topmost left alphanumeric character position of a screen is line 1, column 1. The LINE NUMBER clause of a screen item specifies the line number of the data item, and the COLUMN NUMBER clause specifies the column number.

The screen description entry can also define the following attributes for data on the screen: input-output attribute, display color, brightness, cursor positioning method, and whether the screen is to be erased before display.

In addition to screen items, data items must also be defined before values can be input or output from the screen. The correspondence between screen items and data items is specified in the screen description entry.

2.8.1.2 Screen Item

There are two types of screen items: elementary screen items and group screen items. An area from one column to another column in a line is an elementary screen item. An area containing several elementary screen items is a group screen item. Several group screen items can be grouped together and defined as a group screen item.

An elementary screen item is the smallest unit of screen items that can be processed in one ACCEPT or DISPLAY statement. When a group screen item is specified in an ACCEPT or DISPLAY statement, all subordinate elementary screen items are processed at the same time.

There are four types of elementary screen items:

- Literal item
- Input item
- Output item
- Update item

Literal Item

A literal item is a screen area for displaying a fixed value on the screen. Specify the value of the literal item in the VALUE clause of the screen description entry. Executing a DISPLAY statement displays the value specified in the VALUE clause.

Input Item

An input item is a screen area used to accept data from the screen. Associate one data item with one screen input item in a screen description entry. Execution of an ACCEPT statement moves the input data from the screen to the data item associated with the screen input item. A screen input item cannot display data.

Output Item

An output item is a screen area to display data on the screen. Associate one data item with one screen output item in a screen description entry. Execution of a DISPLAY statement displays the contents of the data item associated with the screen output item on the display. A screen output item cannot accept input from the screen.

Update Item

An update item is a screen area used to obtain input from and display data upon the screen. Associate one data item with one screen update item in screen description entry. Execution of a DISPLAY statement will display the contents of the data item associated with the screen update item. Execution of an ACCEPT statement moves data input from the screen to the data item associated with the screen update item. The value of the data item before it is updated is not maintained.

2.8.2 Input-Output Handling of the Screen

Execute a DISPLAY statement to display the screen. Execute an ACCEPT statement to input data from the screen.

Both elementary and group screen items can be specified in a DISPLAY or ACCEPT statement. Specifying an elementary screen item causes input-output handling to be performed for the elementary screen item only. Specifying a group screen item causes input-output handling to be performed on all the elementary group items subordinate to the group screen item.

Executing an ACCEPT statement moves input data from the screen to the data item associated with the screen input item or screen update item.

Executing a DISPLAY statement displays the value in the data item associated with the screen output item, screen update item or the value of the screen literal item to be displayed on the screen. If the screen item specifies the BLANK or ERASE clause, the area of the screen item is erased before being displayed, otherwise the screen area is not erased.

2.8.3 Screen Input Status

A screen input status is a three byte character area used to indicate the result of an execution of an ACCEPT statement. The value of the screen input status is determined before any unconditional statements associated with the ACCEPT statement are.

The screen input status is specified in the CRT STATUS clause of the SPECIAL-NAMES paragraph.

See the "CRT STATUS Clause" topic in "Configuration Section" section of Chapter 4, the "Environment Division" for a list of the screen input status codes.

F

2.9 Command Line Argument and Environment Variable Modules

There are two modules for handling command line arguments and processing environment variables:

- A module for referencing the number of arguments specified on the command line and the values of the arguments.

A command line is the command line that invoked the COBOL run unit.

- A module that references and updates environment variables.

ACCEPT and DISPLAY statements are used to access the command line and to manipulate the environment variables.

When using ACCEPT or DISPLAY statements, you must specify a mnemonic-name associated with the relevant function-name to access the command line or environment variables.

Use the following descriptions as required in a program which uses the command line argument or environment variable modules.

Program Description	Main Facility
ENVIRONMENT DIVISION	
CONFIGURATION SECTION	
SPECIAL-NAMES paragraph	Associates a function-name with a mnemonic name.
PROCEDURE DIVISION	
Procedure portion	
ACCEPT	Obtains the number of arguments in the command line and their values. Obtains the value of the environment variable.
DISPLAY	Positions arguments in the command line. Positions the environment variable.

2.9.1 Processing Command Line Arguments

A parameter to be passed to a COBOL program, an external switch, or an execution time option can be specified as an argument on the command line. The number of arguments and their values can be determined using the command line argument module.

2.9.1.1 Method for Finding the Number of Command Line Arguments

The ACCEPT statement is used to determine the number of arguments specified on the command line, as well as their values. The ACCEPT statement must include the FROM phrase. The FROM phrase must specify a mnemonic-name. The mnemonic name must be associated with the ARGUMENT-NUMBER function name. Executing such an ACCEPT statement returns the number of arguments on the command line into the identifier associated with the ACCEPT statement.

2.9.1.2 Method for Obtaining the Value of Individual Command Line Arguments

There are two methods for obtaining the value of command line arguments:

- Obtaining the values in sequence
- Specifying the position of an individual argument and obtaining the value

Determine the argument positions to be referenced using the argument position indicator.

Argument Position Indicator

An argument position indicator is an indicator provided in the COBOL concept for specifying the command line argument to be referenced next.

Each argument specified on the command line is given a number in sequence (from left to right) where the number references the argument. The sequence number of the argument is known as the argument position indicator. The initial value of the argument number is 1 and it is incremented by 1 for each additional argument. The initial value of the argument position indicator represents the total number of arguments on the command line.

The value of the argument position indicator is an integer between 0 and 99. The value 0 corresponds to the command (the path and filename of the EXE) used to execute the program.

The value of the argument position indicator is accessed the ACCEPT and DISPLAY statements. These statements must reference a mnemonic-name that is associated with the function-name ARGUMENT-NUMBER or ARGUMENT-VALUE. The initial value of the argument position indicator is 1.

Referencing the Arguments Sequentially

Execute the ACCEPT statement repeatedly to reference the arguments sequentially. Write the mnemonic-name associated with the function-name ARGUMENT-VALUE in the FROM phrase of the ACCEPT statement.

Each time the ACCEPT statement is executed, the value of the argument in the position indicated by the current argument position indicator is retrieved and put in the identifier of the ACCEPT statement. The value of the argument position indicator is incremented by one each time the ACCEPT statement is executed.

Specifying the Position of the Arguments and Then Referencing the Values

To specify the position of an argument and then reference the value, execute a DISPLAY statement and ACCEPT statement in that order.

Specify the position of the argument to be referenced in the identifier or literal of the DISPLAY statement. Then, specify the mnemonic-name associated with the function-name ARGUMENT-NUMBER in the UPON phrase. Executing the DISPLAY statement sets the value specified in the identifier or literal of the DISPLAY statement in the argument position indicator.

Use the mnemonic-name associated with the function-name ARGUMENT-VALUE in the FROM phrase of the ACCEPT statement. Executing the ACCEPT statement obtains the value of the argument (as specified by the current argument position indicator in the identifier of the ACCEPT statement). The current argument position indicator is then incremented by 1.

2.9.2 Accessing an Environment Variable

The value of an environment variable can be referenced and updated using the environment variable operation module.

2.9.2.1 Accessing the Value of an Environment Variable

To access the value of an environment variable, execute a DISPLAY statement and ACCEPT statement in that order.

Specify the environment variable name to be referenced in the identifier or literal of the DISPLAY statement. Then, use the mnemonic-name associated with the function-name ENVIRONMENT-NAME in the UPON phrase. Executing the DISPLAY statement enables input-output operation of the environment variable whose name has been specified in the identifier or literal of the DISPLAY statement.

Use a mnemonic-name associated with the function-name ENVIRONMENT-VALUE in the FROM phrase of the ACCEPT statement. Executing the ACCEPT statement will obtain the value of the environment variable.

2.9.2.2 Updating the Value of an Environment Variable

To update the value of an environment variable, execute two DISPLAY statements.

In the first DISPLAY statement, specify the name of the environment variable to be updated in the identifier or literal and specify the mnemonic-name associated with the function-name ENVIRONMENT-NAME in the UPON phrase. Executing the DISPLAY statement enables input-output operation of the environment variable specified in the identifier or literal of the DISPLAY statement.

In the second DISPLAY statement, specify the updated environment variable value in the identifier or literal and specify a mnemonic-name that is associated with the function-name ENVIRONMENT-VALUE in the UPON phrase. Executing the DISPLAY statement causes the environment variable to be updated.

2.10 Report Writer Module

A report writer module simplifies the procedure for writing reports by defining the report format in the DATA DIVISION. The report writer control system automatically performs processing necessary for determining the columns of the report. This eliminates the need to write code processing things such as the page number count, number of lines in a page, composition of a print line, and line feed in the PROCEDURE DIVISION.

Note

The report writer module cannot be used in [Winx64], [LinuxIPF], [Linux64] and [.NET].

The file that outputs the report is called a "report file." The report file is treated in the same way as a print file in the sequential I-O module. The report file is defined in a DATA DIVISION file description entry.

Several report styles can be output to a single report file. Associate one page style with one report style, and define the report style in the report description entry of the DATA DIVISION. The relationship between a report file and a report description entry is defined in the file control entry REPORT clause.

Several data items can be output to a report. The data to be output to the report is defined in a report group description entry. The report group description entry is a hierarchical structure used to define the data items that compose the report.

Use the following descriptions as required in a program using a report writer module.

Program Description	Main Facility
ENVIRONMENT DIVISION	
INPUT-OUTPUT SECTION	
FILE-CONTROL paragraph	
File control entry	Associates the file-name of a report file with an external medium. Defines the file organization and access mode.
I-O CONTROL paragraph	Specifies the storage area to be shared by several files containing report files.
DATA DIVISION	
FILE SECTION	
File description entry	Defines the physical structure of a report file. Associates the report file with a report-name.
REPORT SECTION	
Report description entry	Defines the physical structure of a report.
Report group description entry	Defines the items in a report.
PROCEDURE DIVISION	
Declarative	
USE statement and USE AFTER STANDARD EXCEPTION	Define the input-output error procedure.
USE BEFORE REPORTING statement and USE BEFORE REPORTING procedure	Define the procedure to be executed before the report group is displayed.
Procedure portion	
OPEN	Opens a file.
INITIALIZE	Starts report processing.
GENERATE	Writes a report.
SUPPRESS	Suppresses display of a report group.
TERMINATE	Ends report processing.
CLOSE	Closes a file.

2.10.1 Report File

A report file is a sequentially organized output file. The report file is defined in a report file description entry of the DATA DIVISION that specifies the REPORT clause. The REPORT clause associates the report file with the report description entry by specifying the report-name of the report description entry in the REPORT clause.

A report file record description entry is not specified in the FILE SECTION as it is when defining a sequential file. Rather, the record structure of a report file is defined in a REPORT SECTION report group description entry.

A report file can be referenced using the OPEN, GENERATE, INITIATE, SUPPRESS, TERMINATE, USE AFTER STANDARD EXCEPTION, USE BEFORE REPORTING, or CLOSE statements.

2.10.2 Report Writer Special Registers

The following two special registers are created by the COBOL runtime system for report files:

- PAGE-COUNTER
- LINE-COUNTER

One of each of these special registers is created for each report defined in the REPORT SECTION. The special register PAGE-COUNTER is called a "page counter," and the special register LINE-COUNTER is called a "line-counter."

2.10.2.1 Page Counter

The page counter is treated as a numeric data item defined as "PIC 9(6) USAGE IS PACKED-DECIMAL." The range of values of the page counter is 1 to 999999. The page counter value is controlled by the report writer control system and can be used in a program to add report page numbers. The page counter can only be referenced in the SOURCE clause of the REPORT SECTION or by statements in the PROCEDURE DIVISION. The page counter value can be manipulated and updated by a program in the PROCEDURE DIVISION.

2.10.2.2 Line-Counter

The line-counter is treated as a numeric data item defined as "PIC 9(6) USAGE IS PACKED-DECIMAL". The range of values of the line-counter is 1 to 999999. The line-counter value is controlled by the report writer control system and can be used to determine a position within the body of a report. Only the SOURCE clause of the REPORT SECTION or a statement in the PROCEDURE DIVISION can reference the line-counter. The line-counter value can be updated by the program's PROCEDURE DIVISION.

Chapter 3 Identification Division and End Program Header

The IDENTIFICATION DIVISION specifies the name of the program and the program attributes. The IDENTIFICATION DIVISION must be placed at the beginning of the program.

The END PROGRAM header indicates the end of the program.

3.1 Composition of the Identification Division

The format of the IDENTIFICATION DIVISION is shown below.

Format

```
[IDENTIFICATION DIVISION.]
PROGRAM-ID. program identification-entry   ] Program name paragraph
[AUTHOR. [comment entry]...].              ] AUTHOR paragraph
[INSTALLATION. [comment entry]...].        ] INSTALLATION paragraph      } IDENTIFICATION
[DATE-WRITTEN. [comment entry]...].        ] DATE-WRITTEN paragraph      } DIVISION
[DATE-COMPILED. [comment entry]...].       ] Compile date paragraph      }
[SECURITY. [comment entry]...].            ] SECURITY paragraph          }
```

Syntax Rules

1. The PROGRAM-ID paragraph must be the first paragraph in the IDENTIFICATION DIVISION. The program name is specified in the PROGRAM-ID paragraph.
2. The program name paragraph can be followed by the AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, or SECURITY paragraphs.

F

The COBOL compiler allows these paragraphs to be specified in any sequence.

These paragraphs are obsolete language elements and are treated as comment lines.

3.1.1 Program-Id Paragraph

This paragraph specifies the name of the program.

Format

```
PROGRAM-ID.  { program-name  [AS literal-1] }
             { program-name-literal         }

[IS { { | COMMON    | } } PROGRAM ].
    { { | INITIAL   | } }
    {   RECURSIVE      }
```

F

The AS phrase function is available for [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

The RECURSIVE clause is available for [Winx64], [Linux64] and [.NET] .

Syntax Rules

1. Specify the name of the program in program-name or program-name-literal(*).

   The name of a nested program cannot be the same as the name of the program which directly or indirectly contains the nested program.

2. The COMMON clause can be written in a nested program only.
3. Program-name must conform to the rules for user-defined words.
4. See the topic titled "Literal for Special Applications" in the "Basic Overview of the Language" section of Chapter 1, "General Rules" for the rules for describing program-name-literal(*).

F

5. Literal-1 must be a nonnumeric literal or a national nonnumeric literal and cannot be a figurative constant.
6. Literal-1 must not be specified in a program that is contained within another program.
7. The RECURSIVE clause can be written in an external program only.
8. When the program that contains this program directly or indirectly is a recursive program, the INITIAL clause cannot be specified.

F

* : Extended functions or functions specific to NetCOBOL.

- program-name-literal

General Rules

1. The character-string specified in the following identifies the source program, object program, and all printed outputs.
   - program name

F

   - program-name-literal
2. Specify the COMMON clause to give the common attribute to a nested program. A program having the common attribute is called a "common program." A common program can be called from programs other than the program containing the common program.
3. Specify the INITIAL clause to give the initial attribute to a program. A program having the initial attribute is called an "initial program." Accessing an initial program activates the initial program and all other programs contained in the initial program.

F

4. When an ENTRY statement specifying a secondary entry point name has been specified in an initial program, accessing the secondary entry point name also initializes the initial program and all other programs contained in the initial program.
5. Program-name names a program that is declared by the following program definition. When literal-1 is specified, the program name becomes both an internal name and an external name. When literal-1 is not specified, the program name becomes an internal and external name.
6. When the RECURSIVE clause is specified, all nested programs inherit the recursive attribute. The program can call itself and it can be called when in the active state. However, nested programs cannot be called directly from other nested programs. Unless the RECURSIVE clause is inherited or specified directly, a program cannot be called again when it is already active.

3.1.2 DATE-COMPILED Paragraph

The DATE-COMPILED paragraph records the date the program was compiled. This paragraph is an obsolete language element.

Format

DATE-COMPILED. [comment-entry]...

Note

In [.NET], the DATE-COMPILED paragraph is treated as a comment line.

General Rules

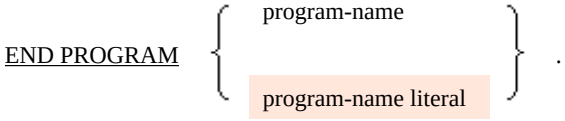
If the DATE-COMPILED paragraph is specified, the compiler will insert the date that the program was compiled in the source program output listing at compilation. The DATE COMPILED paragraph is replaced by the following in the source program list.

```
DATE-COMPILED. compile-date.
```

3.2 End Program Header

The end program header specifies the end of a program.

Format



Syntax Rules

1. Specify the name of the program in the following:

   - program-name

F - program-name-literal

   The name of the program must be the same as the name of the program specified in the first PROGRAM-ID paragraph.

2. The PROGRAM-ID paragraph and END PROGRAM header must be written in pairs. For example, if program A contains program B, the END PROGRAM header for program B must be specified before the END PROGRAM header for program A.

3. Program-name must conform to the rules for user-defined words.

4. See the topic titled "Literal for Special Applications" in the "Basic Overview of the Language" section of Chapter 1, "General Rules" for the rules for describing program-name-literal(*).

5. Program-name names the program that is declared by the following program definition. When literal-1 is specified, program-name becomes the internal name. When literal-1 is not specified, program-name becomes both the internal and external name.

General Rules

1. Specify the END PROGRAM header at the end of the program.

2. The END PROGRAM statement must be specified for every nested program.

3. One of the following must be written immediately after the END PROGRAM header in a nested program:

   a. A header for the IDENTIFICATION DIVISION of another nested program

   b. An END PROGRAM header for the program containing the nested program

4. The END PROGRAM statement must be specified according to the following rules when writing two or more separately compiled programs in succession:

   a. All separately compiled programs must specify the END PROGRAM header, indicating the end of the separately compiled program.

b. The END PROGRAM header indicating the end of the last separately compiled program can be omitted.

Chapter 4 Environment Division

The ENVIRONMENT DIVISION paragraph specifies the environment suited to the characteristics of the computer that runs the program. The following information is specified in the ENVIRONMENT DIVISION:

- Definition of character set
- Definition of collating sequence
- Association of function-name and mnemonic-name
- Definition of symbolic-character, and alphabet-name
- F - Definition of symbolic-constant
- Association of file to an external medium

The ENVIRONMENT DIVISION follows the IDENTIFICATION DIVISION. The ENVIRONMENT DIVISION is optional and may be omitted.

4.1 Composition of the Environment Division

The ENVIRONMENT DIVISION consists of the CONFIGURATION SECTION and the INPUT-OUTPUT SECTION.

The layout of the ENVIRONMENT DIVISION is shown below. The sections and paragraphs making up the ENVIRONMENT DIVISION must be written in the sequence shown.

Format

ENVIRONMENT DIVISION

[CONFIGURATION SECTION.

[SOURCE-COMPUTER. [source-computer-entry]] }Source computer paragraph

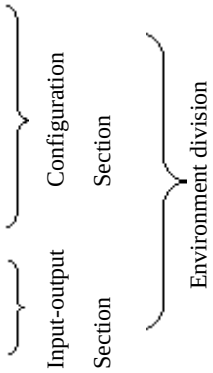
[OBJECT-COMPUTER. [object-computer-entry]] }Object computer paragraph

[SPECIAL-NAMES. [special-names-entry]]] }Special names paragraph

[INPUT-OUTPUT SECTION.

FILE-CONTROL. {file-entry} ... }File control paragraph

[I-O-CONTROL. [I-O-CONTROL-entry]]] }I-O-control paragraph



4.2 CONFIGURATION SECTION

The characteristics of the source and object computer are specified in the CONFIGURATION SECTION.

The CONFIGURATION SECTION can only be written in the outermost program of a program that contains nested programs. It cannot be used in nested programs.

4.2.1 SOURCE-COMPUTER Paragraph

Specify the computer used to compile the program in the SOURCE-COMPUTER paragraph.

Format

SOURCE-COMPUTER.

[computer-name [WITH DEBUGGING MODE clause].]

Syntax Rules

See the topic titled "System-name" in the section "Basic Overview of the Language" of Chapter 1, "General Rules" for the rules for describing computer-name.

General Rules

1. All clauses in the SOURCE-COMPUTER paragraph apply to the program which explicitly or implicitly specified the clause and to all nested programs contained therein.
2. When a SOURCE-COMPUTER paragraph has not been specified in the outermost program of a program that contains nested programs, or in any of the nested programs, the system assumes that the source computer will compile the source program.
3. When the paragraph name of the SOURCE-COMPUTER paragraph is specified but source-computer-entry is omitted, the system assumes that the source computer will compile the source program.

4.2.1.1 WITH DEBUGGING MODE Clause

This clause specifies whether debugging code is compiled or not.

Format

WITH DEBUGGING MODE

General Rules

1. When a WITH DEBUGGING MODE clause has been specified, all debugging lines in the program and in all contained nested programs are compiled exactly as they are written.
2. When a WITH DEBUGGING MODE clause has not been specified in the outermost program or in any contained nested programs, all debugging lines in the source program are regarded as comment lines.
3. Any COPY or REPLACE statements are processed before the WITH DEBUGGING MODE clause is checked.

4.2.2 OBJECT-COMPUTER Paragraph

Specifies the computer to be used to run the program.

Format

OBJECT-COMPUTER.

[computer-name [MEMORY SIZE clause]

[PROGRAM COLLATING SEQUENCE clause]

Computer-name is omissible for [Winx64].

Syntax Rules

See the topic titled "System-name" in the section "Basic Overview of the Language" of Chapter 1, "General Rules" for the rules for describing computer-name.

General Rules

All clauses in the OBJECT-COMPUTER paragraph apply to both the program which explicitly or implicitly specified the clause and to all nested programs it contains.

4.2.2.1 MEMORY SIZE Clause

This clause specifies information relating to storage capacity. It is an obsolete language element.

Format

MEMORY SIZE interger-1 { WORDS / CHARACTERS / MODULES }

F The COBOL compiler regards the MEMORY SIZE clause as a comment.

4.2.2.2 PROGRAM COLLATING SEQUENCE Clause

This clause specifies the collating sequence.

Refer to "PROGRAM COLLATING SEQUENCE Clause" in the "COBOL Syntax Samples", for an example of the usage.

Format

PROGRAM COLLATING SEQUENCE IS alphabet-name-1

Syntax Rules

Alphabet-name-1 must be associated with the collating sequence in the ALPHABET clause of the SPECIAL-NAMES paragraph.

F Alphabet-name-1 must not be the alphabet-name associated with any of the following in the ALPHABET clause.

- function-name
- SJIS
- UTF8
- UTF16
- UTF16LE
- UTF16BE
- UTF32
- UTF32LE
- UTF32BE

Specifying SJIS, UTF8, UTF16, UTF16LE, UTF16BE, UTF32, UTF32LE or UTF32BE in the ALPHABET clause is specific to [Win32], [Winx64] and [Linux64] .

General Rules

1. When a PROGRAM COLLATING SEQUENCE clause has been written, the collating sequence associated with alphabet-name-1 is adopted as the collating sequence for the program.
2. When the PROGRAM COLLATING SEQUENCE clause has been omitted, the native collating sequence is adopted as the collating sequence for the program.
3. If a character relation (excluding a national character relation(*)) is performed in any of the following places, the collating sequence specified in the PROGRAM COLLATING SEQUENCE clause is used to determine the truth value of the condition:
    a. Relation condition specified explicitly
    b. condition-name condition specified explicitly
    c. CONTROL clause in a report description entry specified implicitly

4. When the COLLATING SEQUENCE specification has been omitted from a SORT or MERGE statement, the collating sequence specified in the PROGRAM COLLATING SEQUENCE clause applies to the SORT and MERGE statements.

F 5. The collating sequence of characters specified in the PROGRAM COLLATING SEQUENCE clause does not affect the collating sequence of national characters.

F *: Extended functions or functions specific to NetCOBOL.

- national character relation

4.2.3 SPECIAL-NAMES Paragraph

The symbolic-constant(*), symbolic-character, alphabet-name, character set, and collating sequence is defined in the SPECIAL-NAMES paragraph. The SPECIAL-NAMES paragraph also associates mnemonic-names with function-names.

F *: Extended functions or functions specific to NetCOBOL.

- symbolic-constant

Format

SPECIAL-NAMES.
[[{function-name-1 clause} ...]
[{function-name-2 clause} ...]
[{function-name-3 clause} ...]
[{ALPHABET clause} ...]
[{CLASS clause} ...]
[CRT STATUS clause]
[CURRENCY SIGN clause]
[CURSOR clause]
[DECIMAL-POINT IS COMMA clause]
[{POSITIONING UNIT clause} ...]
[{PRINTING MODE clause} ...]
[{SYMBOLIC CHARACTERS clause} ...]
[{SYMBOLIC CONSTANT clause} ...].]

General Rules

All clauses in the SPECIAL-NAMES paragraph apply to the program which explicitly or implicitly specified the clause and to all nested programs it contains.

The CRT STATUS and CURSOR clauses are not permitted under .NET COBOL.

4.2.3.1 Function-name-1 Clause

This clause associates a function-name that relates to the input and output of data to a mnemonic-name.

Format

function-name-1 IS mnemonic-name-1

Syntax Rules

1. When mnemonic-name-1 is used in the FROM phrase of the ACCEPT statement or the UPON phrase of the DISPLAY statement, function-name-1 must be one of the following:

   - SYSOUT
   - CONSOLE
   - SYSIN
   - SYSERR
   - SYSPUNCH

   SYSOUT and SYSPUNCH are regarded as being synonymous.

2. When specifying mnemonic-name-1 in the BEFORE ADVANCING or the AFTER ADVANCING phrase of a WRITE statement, function-name-1 must be one of the following:

   - CHANNEL-01
   - CHANNEL-02, ... , CHANNEL-12
   - SLC
   - CTL
   - STACKER-01, STACKER-02

   Each of CHANNEL-02 to CHANNEL-12, STACKER-01, and STACKER-02 is regarded as being synonymous with CHANNEL-01.

F

3. When specifying mnemonic-name-1 in the CHARACTER TYPE clause, function-name-1 must be one of the following:

   - HSC
   - F0202
   - H0202
   - F0102
   - F0201
     YX-7P,
     YX-7P-12,YX-7P-21,YX-7P-22,
     YX-7P-H,
     YX-7P-12-H,YX-7P-21-H,YX-7P-22-H,
     YX-9P,
     YX-9P-12,YX-9P-21,YX-9P-22,
     YX-9P-H,
     YX-9P-12-H,YX-9P-21-H,YX-9P-22-H,
     YX-12P,
     YX-12P-12,YX-12P-21,YX-12P-22,
     YX-12P-H,
     YX-12P-12-H,YX-12P-21-H,YX-12P-22-H,
     YA,YA-12,YA-21,YA-22,
     YA-H,YA-12-H,YA-21-H,
     YA-22-H,
     YB,YB-12,YB-21,YB-22,
     YB-H,YB-12-H,YB-21-H,
     YB-22-H,
     YC,YC-12,YC-21,YC-22,
     YC-H,YC-12-H,YC-21-H,
     YC-22-H,
     YD-9P,YD-9P-12,
     YD-9P-21,YD-9P-22,

YD-9P-H,YD-9P-12-H,
YD-9P-21-H,YD-9P-22-H,
YD-12P,YD-12P-12,
YD-12P-21,YD-12P-22,
YD-12P-H,YD-12P-12-H,
YD-12P-21-H,YD-12P-22-H,
TX-7P,TX-7P-12,
TX-7P-21,TX-7P-22,
TX-7P-H,TX-7P-12-H,
TX-7P-21-H,TX-7P-22-H,
TX-9P,TX-9P-12,
TX-9P-21,TX-9P-22,
TX-9P-H,TX-9P-12-H,
TX-9P-21-H,TX-9P-22-H,
TX-12P,TX-12P-12,
TX-12P-21,TX-12P-22,
TX-12P-H,TX-12P-12-H,
TX-12P-21-H,TX-12P-22-H,
TA,TA-12,TA-21,TA-22,
TA-H,TA-12-H,TA-21-H,
TA-22-H,
TB,TB-12,TB-21,TB-22,
TB-H,TB-12-H,TB-21-H,
TB-22-H,
TC,TC-12,TC-21,TC-22,
TC-H,TC-12-H,TC-21-H,
TC-22-H,
TD-9P,TD-9P-12,
TD-9P-21,TD-9P-22,
TD-9P-H,TD-9P-12-H,
TD-9P-21-H,TD-9P-22-H,
TD-12P,TD-12P-12,
TD-12P-21,TD-12P-22,
TD-12P-H,TD-12P-12-H,
TD-12P-21-H,TD-12P-22-H,
GYX-7P,GYX-7P-12,
GYX-7P-21,GYX-7P-22,
GYX-7P-H,GYX-7P-12-H,
GYX-7P-21-H,GYX-7P-22-H,
GYX-9P,GYX-9P-12,
GYX-9P-21,GYX-9P-12-22,
GYX-9P-H,GYX-9P-12-H,
GYX-9P-21-H,GYX-9P-22-H,
GYX-12P,GYX-12P-12,
GYX-12P-21,GYX-12P-12-22,
GYX-12P-H,GYX-12P-12-H,
GYX-12P-21-H,GYX-12P-22-H,
GYA,GYA-12,GYA-21,GYA-22,
GYA-H,GYA-12-H,GYA-21-H,
GYA-22-H,
GYB,GYB-12,GYB-21,GYB-22,
GYB-H,GYB-12-H,GYB-21-H,
GYB-22-H,
GYC,GYC-12,GYC-21,GYC-22,
GYC-H,GYC-12-H,GYC-21-H,
GYC-22-H,
GYD-9P,GYD-9P-12,
GYD-9P-21,GYD-9P-22,

GYD-9P-H,GYD-9P-12-H,
GYD-9P-21-H,GYD-9P-22-H,
GYD-12P,GYD-12P-12,
GYD-12P-21,GYD-12P-22,
GYD-12P-H,GYD-12P-12-H,
GYD-12P-21-H,GYD-12P-22-H,
GTX-7P,GTX-7P-12,
GTX-7P-21,GTX-7P-22,
GTX-7P-H,GTX-7P-12-H,
GTX-7P-21-H,GTX-7P-22-H,
GTX-9P,GTX-9P-12,
GTX-9P-21,GTX-9P-12-22,
GTX-9P-H,GTX-9P-12-H,
GTX-9P-21-H,GTX-9P-22-H,
GTX-12P,GTX-12P-12,
GTX-12P-21,GTX-12P-12-22,
GTX-12P-H,GTX-12P-12-H,
GTX-12P-21-H,GTX-12P-22-H,
GTA-H,GTA-12-H,GTA-21-H,
GTA-22-H,
GTB,GTB-12,GTB-21,GTB-22,
GTB-H,GTB-12-H,GTB-21-H,
GTB-22-H,
GTC,GTC-12,GTC-21,GTC-22,
GTC-H,GTC-12-H,GTC-21-H,
GTC-22-H,
GTD-9P,GTD-9P-12,
GTD-9P-21,GTD-9P-22,
GTD-9P-H,GTD-9P-12-H,
GTD-9P-21-H,GTD-9P-22-H,
GTD-12P,GTD-12P-12,
GTD-12P-21,GTD-12P-22,
GTD-12P-H,GTD-12P-12-H,
GTD-12P-21-H,GTD-12P-22-H

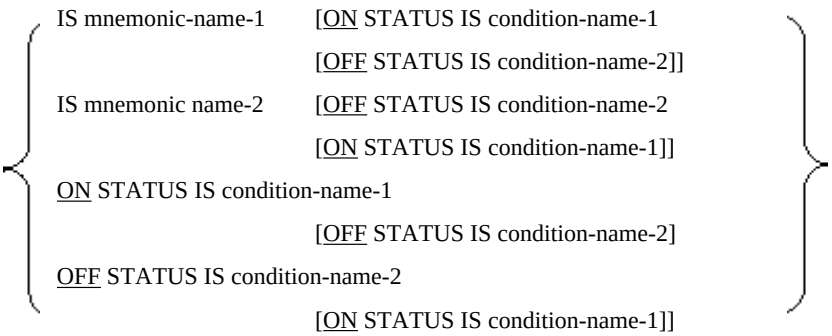
4.2.3.2 Function-name-2 Clause

This clause associates external switches with a mnemonic-name.

Refer to "Function-Name-2 Clause" in the "COBOL Syntax Samples", for an example of the usage.

Format

function-name-2



Syntax Rules

1. Function-name-2 must be SWITCH-n (where n is an integer between 0 and 8).

2. The mnemonic-name associated with function-name-2 can be specified in the SET statement.

General Rules

1. SWITCH-n represents an external switch. SWITCH-0 and SWITCH-8 are synonymous. They represent the same external switch.
2. To check the ON and OFF status of an external switch, associate the two statuses with a condition-name. The IF statement then tests the relevant condition-name to determine whether the switch has been set or not.
3. The status of an external switch can be changed by executing a Format 3 SET statement which contains the mnemonic-name associated with the external switch.
4. A condition-name specified in the SPECIAL-NAMES paragraph can also be referenced from all programs contained within the program.

F

4.2.3.3 Function-name-3 Clause

This clause associates functions used for processing command line arguments and environment variables with a mnemonic-name.

Refer to "Function-name-3 Clause" in the "COBOL Syntax Samples", for an example of the usage.

Format

function-name-3 IS mnemonic-name-1

Syntax Rules

The table below shows the locations in which function-name-3 or the mnemonic-name associated with function-name-3 can be specified.

Name Specified in function-name-3-clause	Location Where a Mnemonic-name Associated with function-name-3 Can Be Written
ARGUMENT-NUMBER	FROM phrase in ACCEPT statement and UPON phrase of DISPLAY statement
ARGUMENT-VALUE	FROM phrase in ACCEPT statement
ENVIRONMENT-NAME	UPON phrase in DISPLAY statement
ENVIRONMENT-VALUE	FROM phrase in ACCEPT statement and UPON phrase in DISPLAY statement

General Rules

1. Use ARGUMENT-NUMBER to retrieve the number of arguments on the command line or to position the argument pointer on the command line.
2. Use ARGUMENT-VALUE to retrieve the value of individual arguments from the command line.
3. Use ENVIRONMENT-NAME to specify the name of an environment variable.
4. Use ENVIRONMENT-VALUE to retrieve the value of an environment variable or to set the value of an environment variable.

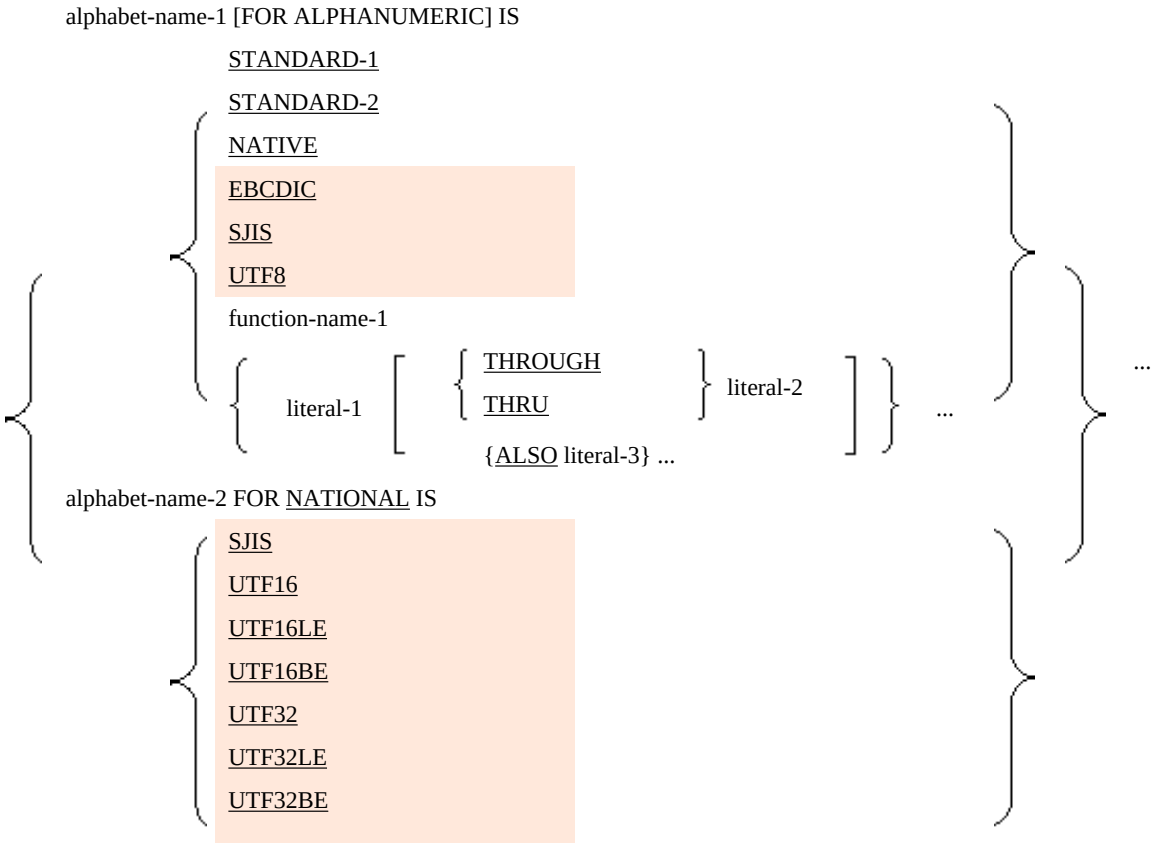
4.2.3.4 ALPHABET Clause

This clause associates an alphabet-name with a character set or the collating sequence.

Refer to "ALPHABET Clause" in the "COBOL Syntax Samples", for an example of the usage.

Format

ALPHABET



SJIS, UTF8 is specific to [Win32], [Winx64] and [Linux64].

NATIONAL phrase is specific to [Win32], [Winx64] and [Linux64].

Syntax Rules

1. Function-name-1 must be CODE-n (where n is an integer between 0 and 9).
2. literal-1 to literal-3 must conform to the following rules:
    a. Any numeric literal specified in literal-1, literal-2, or literal-3 must be an integer between 1 and 256 and must not contain a sign.
    b. When using the THROUGH phrase and specifying a nonnumeric literal in literal-1 and literal-2, the length of literal-1 and literal-2 must be a single character. Similarly, when writing an ALSO phrase and specifying a nonnumeric literal in literal-1 and literal-3, the length of literal-1 and literal-2 must be a single character.
    c. literal-1 to literal-3 must not be a symbolic-character.
    d. literal-1 to literal-3 must not be a national nonnumeric literal(*), Boolean literal(*), or symbolic-constant(*).
3. The same character must not be specified more than once in a literal of a literal phrase.
4. THROUGH and THRU are synonymous.

F

*: Extended functions or functions specific to NetCOBOL.

- national nonnumeric literal
- Boolean literal
- symbolic-constant

General Rules

1. The ALPHABET clause relates a name to a specified code set and/or collating sequence and/or character encoding form. Alphabet-name-1 can be specified in the following locations:

    a. In the PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph

    b. In the COLLATING SEQUENCE of the SORT statement

    c. In the COLLATING SEQUENCE of the MERGE statement

    d. In the CODE-SET clause of a file description entry

    e. In the IN phrase of the SYMBOLIC CHARACTERS clause of the SPECIAL-NAMES paragraph

F f. In the ENCODING clause (specific to [Win32], [Winx64] and [Linux64])

F Alphabet-name-2 can be specified in the ENCODING clause.

2. When STANDARD-1 has been specified, the code set and collating sequence conform to the rules for the standard character set. Each character in the standard character set is associated with a corresponding character in the native character set. The COBOL standard character set is ASCII.

3. When STANDARD-2 has been specified, the code set and collating sequence conform to the rules of the ISO646 international standard. Each character in the standard character set is associated with a corresponding character in the native character set.

F 4. When EBCDIC has been specified, the code set and collating sequence conform to the rules for the EBCDIC code. Each character in the EBCDIC code is associated with a corresponding character in the native character set.

5. When NATIVE has been specified or when the ALPHABET clause has been omitted, the code set and collating sequence conform to the native character set and native collating sequence.

6. CODE-n specified in function-name-1 represents the conversion table. When function-name-1 has been written, the code set and collating sequence conform to the definition in the conversion table.

7. When a literal phrase has been specified, the native code set is assumed as the character alphabet. Define the collating sequence in the literal phrase as follows:

    a. When writing a numeric literal in the literal phrase, specify the sequence numbers of the characters in the native character set in each literal. When writing a nonnumeric literal in the literal phrase, specify the actual characters of the native character set in each literal. When the value of the nonnumeric literal contains several characters, the characters in the literal are regarded as being collated in ascending order starting from the left.

    b. When two or more literal phrases have been written, the collating sequence is ascending in the sequence in which the literal phrases have been written.

    c. Characters belonging to the native character set which are not specified in the literal phrase are placed in a higher sequence position than characters which have been written in the literal phrase. The collating sequence of characters not specified in the literal phrase conform to the native collating sequence.

    d. When the THROUGH phrase has been specified, the collating sequence of all characters which are in literal-1 to literal-2 and which belong to the native character set is ascending in the order literal-1 to literal-2. Characters in a higher sequence position than literal-2 can be specified in literal-1 in the collating sequence for the native character set.

    e. When the ALSO phrase has been written, characters which are indicated by the value of literal-1 and literal-3 are regarded as being in the same sequence position as in the collating sequence. When alphabet-name-1 has been specified in the IN phrase of the SYMBOLIC CHARACTERS clause, only literal-1 is used to indicate the characters in the native character set.

8. The character having the highest sequence position in the collating sequence is assumed as the value of the figurative constant HIGH-VALUE, except when the figurative constant has been specified in literal-1 to literal-3. When there are two or more characters in the highest sequence position, the character which was most recently specified is assumed as the value of the figurative constant HIGH-VALUE.

9. The character having the lowest sequence position in the collating sequence is assumed as the value of the figurative constant LOW-VALUE, except when the figurative constant has been specified in literal-1 to literal-3. When there are two or more characters in the lowest sequence position, the character which was specified first is assumed as the value of the figurative constant LOW-VALUE.

10. When the figurative constants HIGH-VALUE and LOW-VALUE have been specified as the constants in the SPECIAL-NAMES paragraph, HIGH-VALUE and LOW-VALUE represent the highest and lowest characters in the native character set when these constants are referenced.

F

11. When SJIS has been specified, the character encoding form follows the rules of the JIS X 0208 Japanese Industrial Standards.

12. When UTF8, UTF16, UTF16LE, UTFBE, UTF32, UFT32LE, or UTF32BE has been specified, the character encoding form follows the rules of the ISO/IEC 10646 international standard. UTF16LE and UTF32LE are little endian; UTF16BE and UTF32BE are big endian.

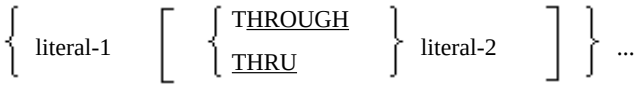
4.2.3.5 CLASS Clause

This clause defines a class-name.

Refer to "CLASS Clause" in the "COBOL Syntax Samples", for an example of the usage.

Format

CLASS class-name-1 IS



Syntax Rules

1. literal-1 and literal-2 must conform to the following rules:

   a. Any numeric literal specified in literal-1 or literal-2 must be an unsigned integer between 1 and 256.

   b. When specifying a THROUGH phrase and specifying a nonnumeric literal in literal-1 or literal-2, the length of each literal must be a single character.

   c. literal-1 and literal-2 must not be a national nonnumeric literal(*), Boolean literal(*), symbolic-character, or symbolic-constant(*).

2. THROUGH and THRU are synonymous.

F

*: Extended functions or functions specific to NetCOBOL.

- national nonnumeric literal
- Boolean literal
- symbolic-constant

General Rules

1. The CLASS clause relates a name to a combination of characters. Class-name-1 can be written only in the class condition.

2. Specify the following literals in literal-1 and literal-2:

   a. When specifying a numeric literal in literal-1 or literal-2, uses the sequence number of the characters in the native character set.

   b. When specifying a nonnumeric literal in literal-1 or literal-2, specify the actual characters of the native character set in each literal. When the value of the nonnumeric literal contains several characters, the characters in the literal are included in the combination of characters which are identified according to the class-name.

c. When the THROUGH phrase has been specified, all characters which are in literal-1 to literal-2 and which belong to the native character set are associated with the combination of characters having the name in class-name-1. The characters in the native character set can also be specified in ascending or descending order depending on the relationship of high and low values in literal-1 and literal-2.

F

4.2.3.6 CRT STATUS Clause

This clause specifies a data item for accepting the screen input status. Note: The CRT STATUS clause is not permitted under .NET COBOL. The screen handling module cannot be used in [Linux], [LinuxIPF], [Linux64] and [.NET].

Format

CRT STATUS IS data-name-1

Syntax Rules

1. Data-name-1 must be a group item defined in the WORKING-STORAGE section or an alphanumeric data item. Data-name-1 must be three character positions long.
2. When data-name-1 is a group item , each character position in the group item must be defined as follows:
   a. The first character position must be a single character alphanumeric data item or an unsigned single column zoned decimal data item.
   b. The second character position must be a single character alphanumeric data item or an unsigned single column zoned decimal data item.
   c. The third character position must be a single character alphanumeric data item.

General Rules

1. The CRT STATUS clause specifies a data item that receives the screen input status of an ACCEPT statement of the screen handling module.
2. The contents of data-name-1 are updated each time an ACCEPT statement is executed.
3. The first character position in data-name-1 is an area for storing screen input status key-1. Status key-1 indicates how the ACCEPT statement was terminated. The table below lists the meanings of screen input status key-1.

Screen Input Status Key-1	Meaning of Screen Input Status Key-1
0	Return from end key or last item
1	Return by user-defined function key
2	Return by the function key of the omitted value, or return by the user-defined function key
9	Error

4. The value "0" in screen input status key-1 indicates that the ACCEPT statement ended normally.
5. The second character position in data-name-1 is an area for storing screen input status key-2. More detailed information about how the ACCEPT statement was terminated is stored in screen input status key-2.

Screen Input Status Key-1	Screen Input Status Key-2	Meaning of Screen Input Status Key-2
0	0	Operator depressed the End key
	1	Return from last item
1	X"00" to X"FF"	Function key number
2	X"00" to X"FF"	Function key number

Screen Input Status Key-1	Screen Input Status Key-2	Meaning of Screen Input Status Key-2
9	X"00"	No relevant item on screen

6. The third position of data-name-1 is used by the COBOL runtime system and should not be referenced.

4.2.3.7 CURRENCY SIGN Clause

This clause specifies the currency symbol.

Refer to "CURRENCY SIGN Clause" in the "COBOL Syntax Samples", for an example of the usage.

Format

CURRENCY SIGN IS literal-1

Syntax Rules

Literal-1 must not be a symbolic-character or symbolic-constant(*).

F

*: Extended functions or functions specific to NetCOBOL.

- symbolic-constant

General Rules

1. literal-1 specifies the currency symbol to be used in the PICTURE clause. literal-1 must be a single character nonnumeric literal. The following characters must not be specified in literal-1:
   - Numeric characters: 0 to 9
   - Uppercase alphabetic characters: A B C D P R S V X Z Lowercase alphabetic characters: a to z Space character
   - Uppercase alphabetic characters: E G N
   - Special characters: * + - , . ; ( ) / = "
2. When the CURRENCY SIGN clause has been omitted, the currency symbol in the COBOL character set is regarded as the currency symbol to be used in the PICTURE clause.

F

4.2.3.8 CURSOR Clause

This clause specifies a data item for specifying the cursor position. Note: The screen handling module cannot be used in [Linux], [LinuxIPF], [Linux64] and [.NET].

Format

CURSOR IS data-name-1

Syntax Rules

1. Data-name-1 must be a data item defined in the WORKING-STORAGE section. There must be four or six character positions in the data item of data-name-1.
2. Data-name-1 must be a group item to which only an unsigned zoned decimal data item or a similar elementary item is subordinate.

General Rules

1. The CURSOR clause specifies a data item for passing the cursor position for an ACCEPT statement of the screen handling module.
2. Move the desired cursor position to data-name-1 before executing the ACCEPT statement.

3. After the ACCEPT statement is executed, the last cursor position used by the user is stored in data-name-1.
4. When the CURSOR clause has been omitted, the cursor is positioned at the beginning of the first input or update item on the screen before the ACCEPT statement is executed.
5. The value stored and returned in data-name-1 have the following meanings:
    a. If data-name-1 is four character positions long, the first two characters represent the line number and the last two characters represent the column.
    b. If data-name-1 is six character positions long, the first three characters represent the line number and the last three characters represent the column.
6. If the line or column values of data-name-1 are not within the character positions for the input or update item on the screen when the ACCEPT statement is executed, the cursor is positioned at the start of the first input or update item on the screen.

4.2.3.9 DECIMAL-POINT IS COMMA Clause

This clause switches the function of the comma and decimal point with each other.

Refer to "DECIMAL-POINT IS COMMA Clause" in the "COBOL Syntax Samples", for an example of the usage.

Format

DECIMAL-POINT IS COMMA

General Rules

The DECIMAL-POINT IS COMMA clause exchanges the function of the comma and decimal point in the character-string and numeric literal of a PICTURE clause.

F

4.2.3.10 POSITIONING UNIT Clause

This clause defines the positioning unit-name.

Refer to "POSITIONING UNIT Clause" in the "COBOL Syntax Samples", for an example of the usage.

Format

POSITIONING UNIT

positioning-unit-name-1 IS literal-1 CPI

Syntax Rules

Literal-1 must be an unsigned numeric literal from 0.01 to 24.00. The number of decimal positions in literal-1 must not exceed 2.

General Rules

1. The POSITIONING UNIT clause specifies a name for the unit which is used when specifying a column in the print line. Positioning-unit-name-1 can be specified in the PRINTING POSITION clause of a data description entry. See the topic titled "PRINTING POSITION clause" of the "Data Description Entry" section of Chapter 5, the "Data Division" for more information.
2. Specify the elementary unit of the positioning unit in literal-1. The unit for literal-1 is CPI (characters per inch).

F

4.2.3.11 PRINTING MODE Clause

This clause defines the printing mode-name.

Refer to "PRINTING MODE Clause" in the "COBOL Syntax Samples", for an example of the usage.

Format

```
PRINTING MODE printing-mode-name-1
    IS [FOR { MOCS | SOCS | ALL } ]
    [IN SIZE literal-1 POINT]
    [AT PITCH literal-2 CPI]
    [WITH FONT function-name-1]
    [AT ANGLE integer-1 DEGREES]
    [BY FORM function-name-2]
```

Syntax Rules

1. Literal-1 must be an unsigned numeric literal from 3.0 to 300.0. The number of decimal positions in literal-1 must not exceed 1.
2. Literal-2 must be an unsigned numeric literal from 0.01 to 24.00. The number of decimal positions in literal-2 must not exceed 2.
3. Function-name-1 must be one of the following:

MINCHOU	Mincho typeface
MINCHOU-HANKAKU	Half-size Mincho typeface
GOTHIC	Gothic typeface
GOTHIC-HANKAKU	Half-size Gothic typeface
GOTHIC-DP	Gothic DP
FONT-nnn	character typeface number established by the system ("nnn" is a typeface number from 001 to 999 established by the system.)

4. Integer-1 must be 0 or 90.
5. Function-name-2 must be one of the following:

F0102	Full-size long typeface
F0201	Full-size flat typeface
F0202	Full-size double-width typeface
H0102	Half-size long typeface
H0201	Half-size flat typeface
H0202	Half-size double-width typeface
H	Half size (En-size)
F	Full size

6. When the FOR phrase has been omitted, FOR MOCS is assumed as the default.
7. When the FONT phrase has been omitted, the following are assumed as the default:
    a. When printing a National data item or National edited data item, WITH FONT MINCHOU is assumed as the default.
    b. When printing an alphabetic data item, alphanumeric data item, alphanumeric edited data item, zoned decimal data item, external Boolean data item, external floating-point data item, or numeric edited data item, WITH FONT GOTHIC-10-LPW is assumed as the default.

8. When the ANGLE phrase has been omitted, AT ANGLE 0 DEGREES is assumed as the default.
9. When the FORM phrase has been omitted, BY FORM F is assumed as the default.

General Rules

1. In the PRINTING MODE clause, supply a printing-mode-name-1 to set the printing mode when printing on the printing device. Printing mode refers to the character size, character pitch, character typeface, character rotation, and character configuration. Printing-mode-name-1 can be specified in the CHARACTER TYPE clause.
2. The FOR phrase specifies the item for which printing mode-name-1 is to be set as the printing mode.
    a. When specifying the printing mode for a national data item or a national edited data item, write the FOR MOCS phrase.
    b. When specifying the printing mode for an alphabetic data item, alphanumeric data item, alphanumeric edited data item, zoned decimal data item, external Boolean data item, external floating-point data item, or numeric edited data item, write the FOR SOCS phrase.
    c. When specifying the printing mode for a national data item, national edited data item, alphabetic data item, alphanumeric data item, alphanumeric edited data item, zoned decimal data item, external Boolean data item, external floating-point data item, or numeric edited data item, use the FOR ALL phrase.
3. Specify the character size in literal-1 of the SIZE phrase. The unit of measure for the character size is a point.
4. Specify the character pitch in literal-2 of the PITCH phrase. The unit of measure for the character pitch is CPI (characters per inch).
5. When either the SIZE or the PITCH phrase has been omitted, the character size and character pitch are set as follows:
    a. When the PITCH phrase has been omitted, a character pitch appropriate for the value in the SIZE phrase is assumed as the value in the PITCH phrase.
    b. When the SIZE phrase has been omitted, a character size appropriate for the value in the PITCH phrase is assumed as the value in the SIZE phrase.
6. When both the SIZE phrase and PITCH phrase have been omitted, the character size and character pitch are assumed as follows:
    a. When printing an alphabetic data item, alphanumeric data item, alphanumeric edited data item, zoned decimal data item, external Boolean data item, external floating-point data item, or numeric edited data item, the character size is set to 7 points and the character pitch is set to 10 CPI.
    b. When printing a national data item or national edited data item, the character size is set to 12 points and the character pitch is set to a value appropriate for the character size.
7. Specify the character typeface in function-name-1 of the FONT phrase.
8. Specify character rotation by specifying the counter-clockwise angle of rotation in integer-1 of the ANGLE phrase. Specifying 0 in integer-1 selects horizontal printing. Specifying 90 in integer-1 selects vertical printing.
9. Specify the character configuration in function-name-1 of the FORM phrase.

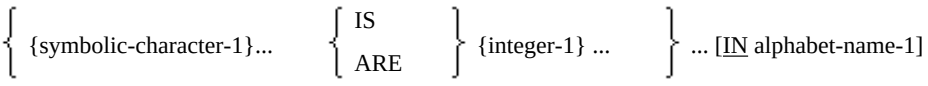
4.2.3.12 SYMBOLIC CHARACTERS Clause

This clause defines symbolic-characters.

Refer to "SYMBOLIC CHARCTERS Clause" in the "COBOL Syntax Samples", for an example of the usage.

Format

SYMBOLIC CHARACTERS



Syntax Rules

1. The same symbolic-character cannot be specified more than once in symbolic-character-1.
2. When specifying lists in symbolic-character-1 and integer-1, specify symbolic-character-1 and integer-1 as a pair. That is, the first symbolic-character-1 and the first integer-1 should make a pair and the second symbolic-character-1 and second integer-1 should also make a pair.
3. The number of lists for symbolic-character-1 and the number of lists for integer-1 must be the same.
4. When omitting the IN phrase, integer-1 must be in the ordinal position of characters in the native character set. When writing the IN phrase, integer-1 must be the ordinal position of characters in the character set specified in alphabet-name-1.
5. Alphabet-name-1 must not be the alphabet-name associated with any of the following in the ALPHABET clause.

   - function-name
   - SJIS
   - UTF8
   - UTF16
   - UTF16LE
   - UTF16BE
   - UTF32
   - UTF32LE
   - UTF32BE

   Specifying SJIS, UTF8, UTF16, UTF16LE, UTF16BE, UTF32, UTF32LE or UTF32BE in the ALPHABET clause is specific to [Win32], [Winx64] and [Linux64].

General Rules

1. In a SYMBOLIC CHARACTERS clause which omits the IN phrase, symbolic-character-1 represents the character whose ordinal position in the native character set is specified by integer-1.
2. In a SYMBOLIC CHARACTERS clause which includes an IN phrase, integer-1 specifies the ordinal position of the character that is represented in the character set named by alphabet-name-2.

F

4.2.3.13 SYMBOLIC CONSTANT Clause

This clause defines a symbolic-constant.

Refer to "SYMBOLIC CONSTANT Clause" in the "COBOL Syntax Samples", for an example of the usage.

Format

SYMBOLIC CONSTANT

{symbolic-constant-1 IS integer-1 }...

Syntax Rules

integer-1 must not be a symbolic-character or a symbolic-constant.

General Rules

1. The SYMBOLIC CONSTANT clause associates a name with a symbolic constant indicated by integer-1.
2. symbolic-constant-1 can be specified in the following locations:

   a. In the literal of the POSITIONING UNIT clause and the literal and integer of the PRINTING MODE clause in the ENVIRONMENT DIVISION.

b. In the integer of the OCCURS clause and the literal of the VALUE clause in the DATA DIVISION.

c. All locations in which a literal can be specified in the PROCEDURE DIVISION.

4.3 Input-Output Section

The INPUT-OUTPUT section is required for transferring data between an external medium and a COBOL program.

4.3.1 File-Control Paragraph

The FILE-CONTROL paragraph associates a file or files with an external medium.

Format

FILE-CONTROL.

{file-control-entry}...

Several file control entries can be specified in a single FILE-CONTROL paragraph. The clauses to be specified in the file-control-entry and the rules for describing the clauses differ depending on the type of file being used.

File-Control Entries for Sequential, Relative, and Indexed Files

The file control entries for sequential, relative, and indexed files define the physical attributes pertaining to each file.

Format 1

Sequential file

SELECT clause
[ACCESS MODE clause]
ASSIGN clause
[FILE STATUS clause]
[FORMAT clause]
[GROUP clause]
[LOCK MODE clause]
[ORGANIZATION clause]
[PADDING CHARACTER clause]
[RECORD DELIMITER clause]
[RESERVE clause].

Format 2

Relative file

SELECT clause
[ACCESS MODE clause]
ASSIGN clause
[FILE STATUS clause]
[LOCK MODE clause]
ORGANIZATION clause
[RESERVE clause].

Format 3

Indexed file

SELECT clause

[ACCESS MODE clause]

[{ALTERNATE RECORD KEY clause}...]

ASSIGN clause

[FILE STATUS clause]

[LOCK MODE clause]

ORGANIZATION clause

RECORD KEY clause

[RESERVE clause].

Syntax Rules

1. The SELECT clause must be specified first for each file control entry. The clauses after the SELECT clause can be written in any sequence.
2. The file-name specified in the file description entry of the DATA DIVISION must be defined once in the FILE-CONTROL paragraph of the same program.

General Rules

1. When the file connector of the file in the SELECT clause is an external file connector, all file control entries in the run unit which reference the same file connector must conform to the following rules:
    a. The OPTIONAL phrase in the SELECT clause must either be specified for all file control entries or omitted for all file control entries.
    F b. When a value other than data-name is specified

    in the ASSIGN clause, the same value must be specified in all file control entries.
    c. The value of the integer specified in the RESERVE clause must be the same for all file control entries.
    d. The same organization must be specified in the ORGANIZATION clause for all file control entries.
    e. The same access mode must be specified in the ACCESS MODE clause for all file control entries.
    f. The same lock mode must be specified in the LOCK MODE clause for all file control entries.
    g. For relative files, the same external data item must be specified as the data-name in the RELATIVE KEY phrase of the ACCESS MODE clause.
    h. File control entries for indexed files must also conform to the following rules:
        - The data-names specified in the RECORD KEY clause must have the same relative position in records relating to the file-name in the SELECT clause. The same value must be defined in all data description entries for the data-names.
        F - The same number of data-names must be specified in the RECORD KEY clause for all file control entries.
        - The DUPLICATES phrase in the RECORD KEY clause must be specified for all indexed files or omitted for all indexed files.
        - The data-names specified in the ALTERNATE RECORD KEY clause must have the same relative position in records relating to the file-name in the SELECT clause. The same value must be defined in all data description entries for the data-names.
        - The same number of data-names must be specified in the ALTERNATE RECORD KEY clause.
        - The DUPLICATES phrase in the ALTERNATE RECORD KEY clause must be specified for all indexed files or omitted for all indexed files.
2. The value of the reference key for an indexed file is related according to the collating sequence of the native character set.

FILE-CONTROL Entry for a Sort-merge File

The FILE-CONTROL entry of a sort-merge file defines the physical attributes of a SORT-MERGE file.

Format

SELECT clause

ASSIGN clause.

Syntax Rules

1. The SELECT clause must be specified first in the file control entry, followed by the ASSIGN clause.
2. The file-name specified in the file description entry of the DATA DIVISION must be defined once in the FILE-CONTROL paragraph of the same program.
3. The COBOL compiler regards an ASSIGN clause for the sort-merge file as a comment.

F

F

FILE-CONTROL Entry for a Presentation File

The FILE-CONTROL entry for a presentation file defines the physical attributes pertaining to a presentation file.

Format

SELECT clause

[ACCESS MODE clause]

ASSIGN clause

[DESTINATION clause]

[END KEY clause]

[FILE STATUS clause]

[FORMAT clause]

[GROUP clause]

[MESSAGE CLASS clause]

[MESSAGE CODE clause]

[MESSAGE MODE clause]

[MESSAGE OWNER clause]

[MESSAGE SEQUENCE clause]

[ORGANIZATION clause]

[PROCESSING CONTROL clause]

[PROCESSING MODE clause]

[PROCESSING TIME clause]

[SELECTED FUNCTION clause]

[SESSION CONTROL clause]

[SYMBOLIC DESTINATION clause]

[UNIT CONTROL clause].

Syntax Rules

1. The SELECT clause must be specified first in the file control entry. The clauses after the SELECT clause can be specified in any sequence.
2. The file-name specified in the file description entry of the DATA DIVISION must be defined once in the FILE-CONTROL paragraph of the same program.
3. The following table lists destination types and indicates, by system, whether clauses can be specified for the destination type.

Clause	System	Destination Type				
		DSP	PRT	ACM	APL	TRM
DESTINATION-1 clause	[DS]	A	A	A	A	A
	[HP]	B	B	B	B	B
	[Solaris]	B	B	A	B	B
	[Win32]	B	B	A	B	B
	[Win16]	B	B	A	E	E
	[Linux]	B	B	E	E	E
	[LinuxIPF]	E	B	E	E	E
	[Linux64]	E	B	E	E	E
	[.NET]	E	B	E	E	E
	[Winx64]	E	B	E	E	E
DESTINATION-2 clause	[DS]	A	A	A	A	A
	[HP]	B	B	B	B	B
	[Solaris]	B	B	A	B	B
	[Win32]	B	B	A	B	B
	[Win16]	B	B	C	E	E
	[Linux]	B	B	E	E	E
	[LinuxIPF]	E	B	E	E	E
	[Linux64]	E	B	E	E	E
	[.NET]	E	B	E	E	E
	[Winx64]	E	B	r	E	E
DESTINATION-3 clause	[DS]	A	A	C	A	A
	[HP]	B	B	C	B	B
	[Solaris]	B	B	C	B	B
	[Win32]	B	B	C	B	B
	[Win16]	B	B	C	E	E
	[Linux]	B	B	E	E	E
	[LinuxIPF]	E	B	E	E	E
	[Linux64]	E	B	E	E	E
	[.NET]	E	B	E	E	E
	[Winx64]	E	B	E	E	E
END KEY clause	[DS]	C	C	C	A	A
	[HP]	C	C	C	B	B
	[Solaris]	C	C	C	B	B
	[Win32]	C	C	C	B	A
	[Win16]	C	C	C	E	E
	[Linux]	C	C	E	E	E
	[LinuxIPF]	E	C	E	E	E
	[Linux64]	E	C	E	E	E
	[.NET]	E	C	E	E	E
	[Winx64]	E	C	E	E	E
FILE STATUS clause	[DS]	A	A	A	A	A
	[HP]	A	A	B	B	B
	[Solaris]	A	A	A	B	A
	[Win32]	A	A	A	B	B
	[Win16]	A	A	A	E	E
	[Linux]	A	A	E	E	E
	[LinuxIPF]	E	A	E	E	E
	[Linux64]	E	A	E	E	E
	[.NET]	E	A	E	E	E
	[Winx64]	E	A	E	E	E
FORMAT clause	[DS]	A	A	C	C	A
	[HP]	A	A	C	C	B
	[Solaris]	A	A	C	C	B

Clause	System	Destination Type				
		DSP	PRT	ACM	APL	TRM
	[Win32]	A	A	C	C	B
	[Win16]	A	A	C	E	E
	[Linux]	A	A	E	E	E
	[LinuxIPF]	E	A	E	E	E
	[Linux64]	E	A	E	E	E
	[.NET]	E	A	E	E	E
	[Winx64]	E	A	E	E	E
GROUP clause	[DS]	A	A	C	C	A
	[HP]	A	A	C	C	B
	[Solaris]	A	A	C	C	B
	[Win32]	A	A	C	C	B
	[Win16]	A	A	C	E	E
	[Linux]	A	A	E	E	E
	[LinuxIPF]	E	A	E	E	E
	[Linux64]	E	A	E	E	E
	[.NET]	E	A	E	E	E
	[Winx64]	E	A	E	E	E
MESSAGE CLASS clause	[DS]	C	C	A	C	C
	[HP]	C	C	B	C	C
	[Solaris]	C	C	A	C	C
	[Win32]	C	C	A	C	C
	[Win16]	C	C	A	E	E
	[Linux]	C	C	E	E	E
	[LinuxIPF]	E	C	E	E	E
	[Linux64]	E	C	E	E	E
	[.NET]	E	C	E	E	E
	[Winx64]	E	C	E	E	E
MESSAGE CODE clause	[DS]	C	C	C	A	C
	[HP]	C	C	C	B	C
	[Solaris]	C	C	C	B	C
	[Win32]	C	C	C	B	C
	[Win16]	C	C	C	E	E
	[Linux]	C	C	E	E	E
	[LinuxIPF]	E	C	E	E	E
	[Linux64]	E	C	E	E	E
	[.NET]	E	C	E	E	E
	[Winx64]	E	C	E	E	E
MESSAGE MODE clause	[DS]	A	A	C	A	A
	[HP]	B	B	C	B	B
	[Solaris]	B	B	C	B	B
	[Win32]	B	B	C	B	B
	[Win16]	C	C	C	E	E
	[Linux]	B	B	E	E	E
	[LinuxIPF]	E	B	E	E	E
	[Linux64]	E	B	E	E	E
	[.NET]	E	B	E	E	E
	[Winx64]	E	B	E	E	E
MESSAGE OWNER clause	[DS]	A	C	C	A	A
	[HP]	B	C	C	B	B
	[Solaris]	B	C	C	B	B
	[Win32]	B	C	C	B	B
	[Win16]	C	C	C	E	E
	[Linux]	B	C	E	E	E

Clause	System	Destination Type				
		DSP	PRT	ACM	APL	TRM
	[LinuxIPF]	E	C	E	E	E
	[Linux64]	E	C	E	E	E
	[.NET]	E	C	E	E	E
	[Winx64]	E	C	E	E	E
MESSAGE SEQUENCE clause	[DS]	D	A	D	D	D
	[HP]	D	B	D	D	D
	[Solaris]	D	B	D	D	D
	[Win32]	D	B	D	D	D
	[Win16]	D	B	D	E	E
	[Linux]	D	B	E	E	E
	[LinuxIPF]	E	B	E	E	E
	[Linux64]	E	B	E	E	E
	[.NET]	E	B	E	E	E
	[Winx64]	E	B	E	E	E
PROCESSING CONTROL clause	[DS]	A	A	C	A	A
	[HP]	B	B	C	B	B
	[Solaris]	B	B	C	B	B
	[Win32]	B	B	C	B	B
	[Win16]	B	B	C	E	E
	[Linux]	B	B	E	E	E
	[LinuxIPF]	E	B	E	E	E
	[Linux64]	E	B	E	E	E
	[.NET]	E	B	E	E	E
	[Winx64]	E	B	E	E	E
PROCESSING MODE clause	[DS]	A	A	A	C	A
	[HP]	A	A	B	C	A
	[Solaris]	A	A	A	C	A
	[Win32]	A	A	A	C	B
	[Win16]	A	A	A	E	E
	[Linux]	A	A	E	E	E
	[LinuxIPF]	E	A	E	E	E
	[Linux64]	E	A	E	E	E
	[.NET]	E	A	E	E	E
	[Winx64]	E	A	E	E	E
PROCESSING TIME clause	[DS]	C	C	A	C	C
	[HP]	C	C	B	C	C
	[Solaris]	C	C	B	C	C
	[Win32]	C	C	A	C	C
	[Win16]	C	C	A	E	E
	[Linux]	C	C	E	E	E
	[LinuxIPF]	E	C	E	E	E
	[Linux64]	E	C	E	E	E
	[.NET]	E	C	E	E	E
	[Winx64]	E	C	E	E	E
SELECTED FUNCTION clause	[DS]	A	A	C	C	A
	[HP]	A	B	C	C	B
	[Solaris]	A	B	C	C	B
	[Win32]	A	B	C	C	B
	[Win16]	A	B	C	E	E
	[Linux]	A	B	E	E	E
	[LinuxIPF]	E	B	E	E	E
	[Linux64]	E	B	E	E	E

Clause	System	Destination Type				
		DSP	PRT	ACM	APL	TRM
	[.NET]	E	B	E	E	E
	[Winx64]	E	B	E	E	E
SESSION CONTROL clause	[DS]	A	A	C	A	A
	[HP]	B	B	C	B	B
	[Solaris]	B	B	C	B	B
	[Win32]	B	B	C	B	B
	[Win16]	B	B	C	E	E
	[Linux]	B	B	E	E	E
	[LinuxIPF]	E	B	E	E	E
	[Linux64]	E	B	E	E	E
	[.NET]	E	B	E	E	E
	[Winx64]	E	B	E	E	E
UNIT CONTROL clause	[DS]	A	A	C	C	C
	[HP]	A	A	C	C	C
	[Solaris]	A	A	C	C	C
	[Win32]	A	A	C	C	C
	[Win16]	A	A	C	E	E
	[Linux]	A	A	E	E	E
	[LinuxIPF]	E	A	E	E	E
	[Linux64]	E	A	E	E	E
	[.NET]	E	A	E	E	E
	[Winx64]	E	A	E	E	E

A: The clause can (or must) be specified, and it is functionally significant.

B: The clause can be specified, but it is not functionally significant. If specified, the clause is treated as a comment.

C: The clause cannot be specified. If specified, the clause is treated as a comment.

D: The clause cannot be specified.

E: Specification of the clause is not significant.

General Rules

When the file connector is an external file connector, all file control entries within a run unit referencing the same file connector must adhere to the following rules:

a. The file-identifier name specified in the ASSIGN clause must be the same.

b. The same ORGANIZATION must be specified in the ORGANIZATION clause.

c. The ACCESS MODE clauses must be identical.

d. The following clauses must have the same specifications.

- SYMBOLIC DESTINATION clause
- DESTINATION-1 clause
- DESTINATION-2 clause
- DESTINATION-3 clause
- END KEY clause
- FORMAT clause
- GROUP clause
- MESSAGE CLASS clause
- MESSAGE CODE clause

- MESSAGE MODE clause
- MESSAGE OWNER clause
- PROCESSING CONTROL clause
- PROCESSING MODE clause
- PROCESSING TIME clause
- SELECTED FUNCTION clause
- SESSION CONTROL clause
- UNIT CONTROL clause

e. The data items specified in following clauses must have the same length.

- MESSAGE CODE clause
- PROCESSING CONTROL clause
- UNIT CONTROL clause

File-Control Entry for a Report Writer File

The FILE-CONTROL entry of a report writer file defines the physical attributes pertaining to a report file.

Format

SELECT clause

[ACCESS MODE clause]

ASSIGN clause

[FILE STATUS clause]

[ORGANIZATION clause]

[RESERVE clause].

Syntax Rules

1. The SELECT clause must be specified first in the FILE-CONTROL entry. The clauses after the SELECT clause can be written in any sequence.
2. The file-name specified in the file description entry of the DATA DIVISION must be defined once in the FILE-CONTROL paragraph of the same program. The REPORT clause must be written in the file description entry of a report file.

General Rules

When the file connector of the file is an external file connector, all FILE-CONTROL entries in a run unit which reference the same file connector must conform to the following rules:

a. The OPTIONAL phrase in the SELECT clause must either be specified for all file control entries or omitted for all file control entries.

b. The file-identifier specified in the ASSIGN clause must be the same for all file control entries.

c. The value of the integer specified in the RESERVE clause must be identical for all FILE-CONTROL entries.

d. The same organization must be specified in the ORGANIZATION clause for all FILE-CONTROL entries.

e. The same access mode must be specified in the ACCESS MODE clause for all FILE-CONTROL entries.

4.3.1.1 ACCESS MODE Clause (Sequential, Relative, Indexed, Presentation(*), and Report Writer Files)

This clause specifies the access mode for a file.

F

*: Extended functions or functions specific to NetCOBOL.

- Presentation

Format 1

Sequential file, presentation file, and report writer

ACCESS MODE IS SEQUENTIAL

Format 2

Relative file

```
ACCESS MODE IS
  { SEQUENTIAL        [RELATIVE KEY IS data-name-1]
  { { RANDOM   }
  { { DYNAMIC  }  RELATIVE KEY IS data-name-1
```

Format 3

Indexed file

```
                  { SEQUENTIAL }
ACCESS MODE IS    { RANDOM     }
                  { DYNAMIC    }
```

Syntax Rules

- Rules for Format 2
  1. Data-name-1 can be qualified.
  2. Data-name-1 must be an unsigned integer item. However, this does not apply if BINARY-CHAR UNSIGNED is specified in the USAGE clause. Do not specify "P" in the PICTURE character-string of the data description entry for data-name-1.
  3. Define data-name-1 in one of the following:
     - WORKING-STORAGE section
     - LINKAGE section
     - LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)
  4. When specifying the file in the USING or GIVING phrase of a SORT or MERGE statement, the ACCESS MODE clause of the file cannot specify RANDOM or DYNAMIC.
  5. When specifying the file in a START statement, the ACCESS MODE clause must specify the RELATIVE KEY phrase.
- Rules for Format 3

  When specifying the file in the USING or GIVING phrase of the SORT or MERGE statement, the ACCESS MODE clause cannot specify RANDOM and DYNAMIC.

General Rules

- Rules common to ALL Formats

  1. ACCESS MODE IS SEQUENTIAL specifies that records are to be accessed sequentially. A file in which specifies ACCESS MODE IS SEQUENTIAL or a file in which omits the ACCESS MODE clause is called a "sequential access mode file."

     ACCESS MODE IS RANDOM specifies that records are to be accessed randomly. A file in which ACCESS MODE IS RANDOM has been specified is called a "random access mode file."

     ACCESS MODE IS DYNAMIC specifies that records are to be accessed dynamically (i.e. records can be accessed sequentially or randomly). A file in which ACCESS MODE IS DYNAMIC has been specified is called a "dynamic access mode file."

  2. When the ACCESS MODE clause has been omitted, ACCESS MODE IS SEQUENTIAL is assumed as the default.

  3. If the file connector is an external file connector, the access mode must be identical for all file control entries in the run unit which reference the same file connector. Additionally, in Format 2 the same external data item must be specified in data-name-1.

- Rules for Format 1

  Records in the file are accessed in same sequence as they were originally written.

- Rules for Format 2

  1. The records in a sequentially accessed file are accessed in ascending order using the relative record numbers of the file.

  2. In a random access mode file, specify the relative record number of the record to be accessed in data-name-1.

  3. The records in a dynamic access mode file can be accessed in either sequential or random access mode.

  4. Records stored in a relative file are each identified by a relative record number. The relative record number indicates the logical position of the record within the file. The relative record number of the first record is 1 and the relative record numbers of the following records are in ascending order (2, 3, 4, ...).

  5. Data-name-1 is used to specify a relative record number to the I-O module.

- Rules for Format 3

  1. In sequential access mode, records are accessed in ascending order (according to the collating sequence of the file), using the key of reference for the file.

  2. In a random access mode file, the record to access is specified in the prime record key item or the alternate record key item.

  3. In dynamic access mode, records can be accessed in either sequential access mode or random access mode.

4.3.1.2 ALTERNATE RECORD KEY Clause (Indexed File)

This clause specifies the alternate record key.

Format

{ALTERNATE RECORD KEY IS

{data-name-1} ... [WITH DUPLICATES]}...

Syntax Rules

1. Data-name-1 can be qualified.

2. Data-name-1 must be defined in the record description entry relating to the file-name specified in the ALTERNATE RECORD KEY clause.

3. The type of data-name-1 depends on the file system. Refer to the section on using other file systems or comparing different file systems in your "NetCOBOL User's Guide" for details of the types supported by your file system.
4. The leftmost character position in the data item data-name-1 must not be the same as either of the following character positions:
    a. The leftmost character position of the prime record key.
    b. The leftmost character position of any alternate record keys within the same file.
5. When the indexed file contains a variable-length record, the first n characters of the variable-length record must contain the alternate record key. Here, n represents the size of the smallest record in the file. See the topic titled "RECORD Clause (Sequential file, Relative file and Indexed file)" of the "File Description Entry" section in Chapter 5, the "Data Division" for more information.
6. Data-name-1 must not be in a variable position in the record.
7. See Appendix B, "System Quantitative Restrictions," for information on the maximum length of data-name-1 and the maximum number of data-name-1 descriptions allowed.

General Rules

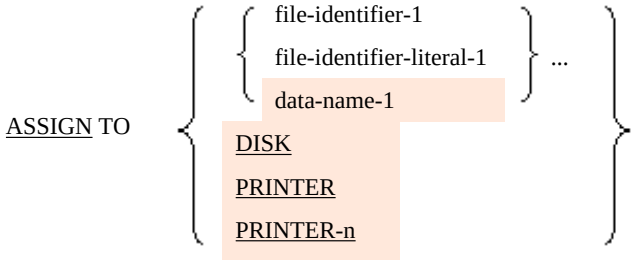
1. The ALTERNATE RECORD KEY clause specifies an alternate record key for the file.
2. The following items must be exactly as they were when the file was created:
    a. Data description entry of data-name-1
    b. Relative position in the record of data-name-1
    c. Number of data-name-1 descriptions
    d. Number of ALTERNATE RECORD KEY clauses
    e. Presence or absence of a DUPLICATES phrase
3. When the DUPLICATES phrase has been specified, the file may contain records having duplicate alternate record key values. When the DUPLICATES phrase has been omitted, the file cannot contain records having duplicate alternate record key values.
4. When associating two or more record description entries with one file, data-name-1 must be written in only one of the record description entries. The same character position as data-name-1 can be referenced implicitly as a key in any of the other record description entries in which data-name-1 has not been written.
5. When the file connector is an external file connector, all file description entries relating to the same file connector in the run unit must conform to the following rules:
    a. Data-name-1 must be in the same relative position in the record for all data description entries.
    b. The contents of the data description entries of data-name-1 must be identical.
    c. The number of data-name-1 descriptions must be identical.
    d. The DUPLICATES phrase must be specified for all data description entries or omitted for all data description entries.
    e. The number of ALTERNATE RECORD KEY clauses must be identical.
6. F When two or more data-name-1 descriptions are specified, they are linked in the sequence in which they were written and treated as a single alternate record key.

4.3.1.3 ASSIGN Clause (Sequential File, Relative File, and Indexed File)

This clause associates a file with an external medium.

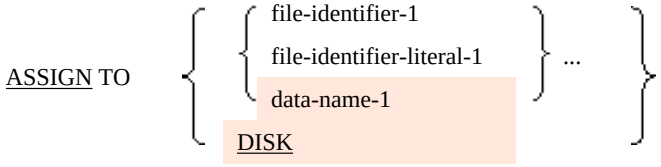
Format 1

Sequential file



Format 2

Relative file and indexed file



F The PRINTER-n phrase is a function specific to [Win16], [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64], and [.NET].

Syntax Rules

F 1. DISK, PRINTER, or PRINTER-n must not be specified as data-name-1.

2. Data-name-1 must be an alphanumeric data item not exceeding 256 characters or a group item.
   Define data-name-1 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
   - LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)

3. The format of file-identifier-1 is shown below.
   - Format 1 : [comment-]name
   - Format 2 : name

   "name" specifies an external file name. "name" must be a character-string not exceeding eight characters consisting of only alphabetic and numeric characters. The first character in "name" must be an alphabetic character.

4. See the topic titled "Literal for Special Applications" in the "Basic Overview of the Language" section of Chapter 1, "General Rules" for rules for describing file-identifier-literal-1.

F 5. DISK, PRINTER, or PRINTER-n cannot be specified in a print file with a FORMAT clause.

6. n for PRINTER-n must be an integer from 1 through 9.

General Rules

1. The ASSIGN clause associates file-name with a file on an external storage medium.

F 2. When more than one file-identifier-1, file-identifier-literal-1, or data-name-1 descriptions have been specified, the second and subsequent descriptions are regarded as comments.

3. File-identifier-1 associates file-name with a physical file. Use file-identifier-1 to indirectly associate file-name with a physical file name on a storage medium.

4. File-identifier-literal-1 associates file-name with the physical file. Use file-identifier-literal-1 to directly associate file-name with a physical file name on a storage medium.

F

5. Data-name-1 associates file-name with the physical file name as indicated by the value of data-name-1. Data-name-1 specifies the value directly representing the physical file name on the storage medium. Data-name-1 must be initialized with the physical file name before an OPEN statement can be executed.
6. The DISK phrase specifies the name of a disk file whose name is the same as file-name.
7. The PRINTER or PRINTER-n phrase associates file-name with a printing device. PRINTER denotes the system's main (default) printer. PRINTER-1 denotes the system's second printer, PRINTER-n denotes the system's "nth" printer.
8. Refer to "NetCOBOL User's Guide" for information on the relationship between the contents of the ASSIGN clause and the physical file.

4.3.1.4 ASSIGN Clause (Sort-merge and Report Writer)

This clause associates a file with an external medium.

Format

ASSIGN TO { file-identifier-1 / file-identifier-literal-1 } ...

Syntax Rules

The format of file-identifier-1 is shown below.

```
[comment-]name
```

Specify an external file name in "name." "name" must be a character-string not exceeding eight characters and consist of only alphabetic and numeric characters. The first character in "name" must be an alphabetic character.

General Rules

1. File-identifier-1 specifies the physical file name of the file to be referenced.

F

2. When more than one file-identifier-1 descriptions has been specified, the second and subsequent descriptions are regarded as comments.
3. The ASSIGN clause of a sort-merge file is treated as a comment.

F

4.3.1.5 ASSIGN Clause (Presentation File)

This clause associates file-name with an external file name.

Format

ASSIGN TO {file-identifier-1} ...

Syntax Rules

The format of file-identifier-1 is shown below.

```
GS-name
```

"name" specifies the external file name of the file to reference. "name" must be a character-string not exceeding eight characters and consist of only alphabetic and numeric characters. The first character in "name" must be an alphabetic character.

General Rules

1. File-identifier-1 specifies the name of the file to access.

2. When more than one file-identifier-1 description has been specified, the second and subsequent descriptions are regarded as comments.

F

4.3.1.6 DESTINATION Clause (Presentation File)

The DESTINATION clause specifies a destination-name.

Format

[DESTINATION-1 IS data-name-1]

[DESTINATION-2 IS data-name-2]

[DESTINATION-3 IS data-name-3]

Syntax Rules

1. Data-name-1, data-name-2, and data-name-3 must be eight-byte alphanumeric data items.
   Define these data items in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
2. Data-name-1, data-name-2, and data-name-3 can be qualified.
3. Data-name-1, data-name-2, and data-name-3 must not be placed in variable locations in a record.

General Rules

1. Data-name-1, data-name-2, and data-name-3 specify message destination names. These destination names are posted to the system when an input-output statement is executed for a presentation file. When a READ statement has been executed successfully, the system sets the destination name.

   The meaning of the DESTINATION clause depends on the system, the input-output statement, and the message type. For details, refer to "NetCOBOL User's Guide".
2. If the DESTINATION clause is omitted, the system uses spaces as the destination name.

F

4.3.1.7 END KEY Clause (Presentation File)

The END KEY clause specifies segment information.

Note

The END KEY clause cannot be used in [.NET].

Format

END KEY IS data-name-1

Syntax Rules

1. Data-name-1 must be a one-character alphanumeric data item.
   Define data-name-1 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
2. Data-name-1 can be qualified.
3. Data-name-1 must not be placed in a variable location in a record.

General Rules

1. The data-name-1 data item specifies segment information (i.e., whether a message is segmented and if so, which segment it is, first, intermediate, or last). When a READ or WRITE statement has been executed successfully, the system determines the segment information.
2. If the END KEY clause is omitted, the system assumes that the message is not segmented when a WRITE statement is executed.

4.3.1.8 FILE STATUS Clause (Sequential File, Relative File, Indexed File, Presentation File(*), and Report Writer)

This clause specifies which data item is to receive the I-O status.

F

*: Extended functions or functions specific to NetCOBOL.

- Presentation File

Format

FILE STATUS IS data-name-1 [data-name-2]

Syntax Rules

1. Data-name-1 and data-name-2(*) can be qualified.
2. Data-name-1 must be a two-character alphanumeric data item or a group item.
   Define data-name-1 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
   - LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)

F

F

3. Data-name-2 must be a four-character alphanumeric data item or a group item.
   Define data-name-2 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
   - LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)
4. Data-name-1 and data-name-2 must not be located in a variable position in a record.
5. Data-name-2 may be written only in the following locations:
   a. The FILE STATUS clause of a print file with a FORMAT clause.
   b. The FILE STATUS clause of a presentation file.

*: Extended functions or functions specific to NetCOBOL.

- data-name-2

General Rules

1. Data-name-1 specifies the location to where the I-O status will be moved after every I-O statement of the file.

F

   Data-name-2 specifies the location to where detailed I-O status information will be stored after every I-O statement on the file.

2. Executing an input-output statement for a file with a FILE STATUS clause causes the I-O status to be moved into data-name-1 and detailed information moved into data-name-2(*).

   Refer to "NetCOBOL User's Guide" for information on the values given as the I-O status and detailed information.

F

*: Extended functions or functions specific to NetCOBOL.

- data-name-2

F

4.3.1.9 FORMAT Clause (Sequential File, Presentation File)

This clause specifies a screen form descriptor.

Format

FORMAT IS data-name-1

Syntax Rules

1. Data-name-1 can be qualified.
2. Data-name-1 must be an eight-character alphanumeric data item or a group item.
   Define data-name-1 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
   - LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)
3. Data-name-1 must not be located in a variable position in a record.
4. When writing a description for a sequential file, the FORMAT clause can be specified only for a print file. A print file for which the FORMAT clause has been specified is called a "print file with FORMAT clause."

General Rules

1. Data-name-1 specifies the name of a screen form descriptor. Executing a WRITE statement for a file in which the FORMAT clause has been specified causes the screen form descriptor specified in data-name-1 to be output to the system.
2. If you explicitly specify that the destination type is a printing device in the SYMBOLIC DESTINATION clause, data-name-1 must be an alphanumeric or group item of a maximum of 30 characters. In other cases, data-name-1 contains a value other than a space, records are edited in accordance with the screen form descriptor specified.
3. Records are not edited when data-name-1 contains a space.
4. When writing a description for a presentation file that omits the FORMAT clause, the screen form descriptor is assumed to contain spaces.
5. Refer to "NetCOBOL User's Guide" for information on the screen form descriptor.

F

4.3.1.10 GROUP Clause (Sequential File, Presentation File)

The GROUP clause indicates a partition name, an item group name or an item name.

Format

GROUP IS data-name-1

Syntax Rules

1. Data-name-1 can be qualified.
2. Data-name-1 must be an eight-character alphanumeric data item or a group item.
   Data-name-1 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section

- LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)

3. Data-name-1 must not be located in a variable position in a record.
4. A GROUP clause can only be specified for files that also specify the FORMAT clause.

General Rules

1. Data-name-1 contains the name of an item name or item group name.
2. When data-name-1 contains a value other than spaces, it specifies the item group name on which to perform I-O.
3. If the GROUP clause has been omitted, the item name or item group name is assumed to be spaces.
4. Refer to "NetCOBOL User's Guide" for information on the item name or item group name.

F 4.3.1.11 LOCK MODE Clause (Sequential File, Relative File, and Indexed File)

This clause specifies the locking mode of a file.

Format 1

Sequential file

```
LOCK MODE IS { AUTOMATIC [WITH LOCK ON RECORD] }
             { EXCLUSIVE                       }
```

Format 2

Relative file or indexed file

```
LOCK MODE IS { MANUAL WITH LOCK ON MULTIPLE RECORDS }
             { AUTOMATIC [WITH LOCK ON RECORD]      }
             { EXCLUSIVE                            }
```

General Rules

- Rules common to Format 1 and Format 2

  1. When the LOCK MODE clause has been omitted, files are opened in exclusive mode unless the file has been opened for input. Files opened in input mode (with no WITH LOCK phrase) are opened in shared mode.
  2. When LOCK MODE IS EXCLUSIVE has been specified, normal execution of an OPEN statement causes the file status for the file to be updated indicating that the file was successfully opened exclusively.
  3. When LOCK MODE IS AUTOMATIC has been specified, normal execution of an OPEN statement in which WITH LOCK is omitted causes the file status for the file to be updated to indicate that the file was opened in shared mode.
  4. The WITH LOCK ON RECORD phrase is regarded as a comment.
  5. A file opened in exclusive mode cannot be opened by another file connector.
  6. A file opened in shared mode can be opened by any other file connector which does not stipulate exclusive mode.
  7. A file which does not exist cannot be opened in shared mode.
  8. Files which are opened in output mode are opened in exclusive mode regardless of whether the LOCK MODE clause has been specified.

- Rules for Format 1

  When LOCK MODE IS AUTOMATIC has been specified, normal execution of a READ statement locks the record. The lock on the record is released upon normal execution of an input-output statement for the file connector which locked the record.

- Rules for Format 2

    1. When LOCK MODE IS MANUAL has been specified, an OPEN statement without the WITH LOCK phrase opens the file in shared mode.
    2. When LOCK MODE IS MANUAL has been specified, the READ WITH LOCK statement locks the record.
    3. When LOCK MODE IS AUTOMATIC has been specified, normal execution of a READ statement locks the record. The lock on the record is released upon normal execution of an input-output statement other than a START statement for the file connector which locked the record.

F

4.3.1.12 MESSAGE CLASS Clause (Presentation File)

The MESSAGE CLASS clause indicates the priority of a message in asynchronous message communication.

The MESSAGE CLASS clause cannot be used in [.NET].

Format

MESSAGE CLASS data-name-1

Syntax Rules

1. Data-name-1 must be a one-digit unsigned external decimal integer item whose picture clause must not contain "P".
2. Define data-name-1 in one of the following:
    - WORKING-STORAGE section
    - LINKAGE section
3. Data-name-1 can be qualified.
4. The data-name-1 data item must not be in a variable location in a record.

General Rules

1. Data-name-1 specifies the priority of a message in asynchronous message communication.
2. The value specified for data-name-1 must be in the range of 1 to 9, with 1 as the highest priority.
3. The value of data-name-1 applies when a message is sent using the WRITE statement. The message is given the processing priority specified in data-name-1.
4. If a message is given a priority when it is sent, the same priority applies when that message is received using the READ statement.
5. Messages with the same priority are received in the order they were sent.
6. If the MESSAGE CLASS clause is omitted, the message is given the lowest priority.

F

4.3.1.13 MESSAGE CODE Clause (Presentation File)

The MESSAGE CODE clause specifies or references a reason code.

The MESSAGE CODE clause cannot be used in [.NET].

Format

MESSAGE CODE IS data-name-1

Syntax Rules

1. Data-name-1 must be an alphanumeric data item, 1 through 32767 characters long, or an unsigned zoned decimal data item of four digits in length.
   Define this data item in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
2. Data-name-1 can be qualified.
3. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. Data-name-1 is used to specify a reason code to forcibly terminate conversational processing. When a READ statement has been executed successfully, the system returns a reason code. The reason code must be updated when a WRITE statement is executed.
2. If the MESSAGE CODE clause is omitted, the reason code is assumed to contain a space.

F

4.3.1.14 MESSAGE MODE Clause (Presentation File)

The MESSAGE MODE clause is used to specify or reference a message type.

Note

For [.NET], the MESSAGE MODE clause is regarded as a comment.

Format

MESSAGE MODE IS data-name-1

Syntax Rules

1. Data-name-1 must be a one-character alphanumeric data item.
   Define this data item in one of following:
   - WORKING-STORAGE section
   - LINKAGE section
2. Data-name-1 can be qualified.
3. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. Data-name-1 specifies a message type (e.g., input message, response message, forced message, transmission result notification request, or forced termination of conversational processing). When a READ statement has been executed successfully, the system returns a message type. The message type must be updated when a WRITE statement is executed.
2. The characters that can be specified to indicate message types depend on the system.

F

4.3.1.15 MESSAGE OWNER Clause (Presentation File)

The MESSAGE OWNER clause is used to specify or reference send authority information.

Note

The MESSAGE OWNER clause cannot be used in [.NET].

Format

MESSAGE OWNER IS data-name-1

Syntax Rules

1. Data-name-1 must be a one-character alphanumeric data item.
   Define this data item in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
2. Data-name-1 can be qualified.
3. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. Data-name-1 contains send authority information. When a READ statement has been executed successfully, the system updates data-name-1 to indicate whether send authority has been granted. When a WRITE statement is executed, data-name-1 must be updated to indicate whether to grant send authority to the program with which conversational processing is performed.
2. If the MESSAGE OWNER clause is omitted, send authority information is assumed to contain spaces.

F

4.3.1.16 MESSAGE SEQUENCE Clause (Presentation File)

The MESSAGE SEQUENCE clause contains the generation serial number of forms.

Note

The MESSAGE SEQUENCE clause cannot be used in [.NET].

Format

MESSAGE SEQUENCE IS data-name-1

Syntax Rules

1. Data-name-1 must be an eight-digit unsigned zoned decimal integer item whose picture string does not contain "P".
2. Define data-name-1 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
3. Data-name-1 can be qualified.
4. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. When a WRITE statement has been executed successfully, data-name-1 is updated with a generation serial number.
2. If execution of a WRITE statement fails, the contents of data-name-1 are undefined.

4.3.1.17 ORGANIZATION Clause (Sequential File)

This clause specifies whether the logical organization of a file is sequential or line sequential(*).

F

*: Extended functions or functions specific to NetCOBOL.

- line sequential

Format

[ORGANIZATION IS] [LINE] SEQUENTIAL

General Rules

1. The ORGANIZATION IS SEQUENTIAL clause specifies that the logical organization of the file is sequential. The file organization is determined at the creation of the file and cannot be changed later.

F

2. The ORGANIZATION IS LINE SEQUENTIAL clause specifies that the logical organization of the file is line sequential. The file organization is determined at the creation of the file and cannot be changed later. The records in a line sequential file are delimited by delimiters. Each record can contain only printable characters and record delimiters. One record is counted as one line.
3. When the ORGANIZATION clause has been omitted, the ORGANIZATION IS SEQUENTIAL clause is assumed as the default.

4.3.1.18 ORGANIZATION Clause (Relative File)

This clause specifies relative organization as the logical organization of a file.

Format

[ORGANIZATION IS] RELATIVE

General Rules

The ORGANIZATION IS RELATIVE clause specifies that the logical organization of the file is relative. The file organization is determined at the creation of the file and cannot be changed later.

4.3.1.19 ORGANIZATION Clause (Indexed File)

This clause specifies indexed organization as the logical organization of a file.

Format

[ORGANIZATION IS] INDEXED

General Rules

The ORGANIZATION IS INDEXED clause specifies that the logical organization of the file is indexed. The file organization is determined at the creation of the file and cannot be changed later.

4.3.1.20 ORGANIZATION Clause (Presentation File(*) and Report Writer)

This clause specifies sequential organization as the logical organization of a file.

F

*: Extended functions or functions specific to NetCOBOL.

- Presentation File

Format

[ORGANIZATION IS] SEQUENTIAL

General Rules

1. The ORGANIZATION IS SEQUENTIAL clause specifies that the logical organization of the file is sequential. The file organization is determined at the creation of the file and cannot be changed later.
2. When the ORGANIZATION IS SEQUENTIAL clause has been omitted, this clause is assumed as the default.

4.3.1.21 PADDING CHARACTER Clause (Sequential File)

This clause specifies a padding character in a block.

Format

PADDING CHARACTER IS { data-name-1 / literal-1 }

F

The COBOL compiler regards the PADDING CHARACTER clause as a comment.

F

4.3.1.22 PROCESSING CONTROL Clause (Presentation File)

The PROCESSING CONTROL clause is used to specify or reference extended message control information.

Format

PROCESSING CONTROL IS data-name-1

Syntax Rules

1. Data-name-1 must be a data item of up to 128 alphanumeric characters.
   Define this data item in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
2. Data-name-1 can be qualified.
3. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. Data-name-1 specifies system-specific message control information. Refer to "NetCOBOL User's Guide" for the format of data-name-1.
2. At the completion of a READ statement, the system writes control information to data-name-1. The value of data-name-1 is must be updated before a WRITE statement is executed.

F

4.3.1.23 PROCESSING MODE Clause (Presentation File)

This clause is used to set the processing type.

Format

PROCESSING MODE IS data-name-1

Syntax Rules

1. Data-name-1 must be a two-character alphanumeric data item or a group item.
   Define data-name-1 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
   - LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)
2. Data-name-1 can be qualified.
3. Data-name-1 must not be located in a variable position in a record.

General Rules

1. Data-name-1 is used to specify the processing type. Processing type refers to the type of input alarm sound and erase all screens input-output.
2. When the PROCESSING MODE clause has been omitted, the processing type is assumed to be spaces.
3. Refer to "NetCOBOL User's Guide" for information on the processing type.

F 4.3.1.24 PROCESSING TIME Clause (Presentation File)

The PROCESSING TIME clause specifies the wait time for message transmission and reception in asynchronous message communication.

Note

The PROCESSING TIME clause cannot be used in [.NET].

Format

PROCESSING TIME data-name-1

Syntax Rules

1. Data-name-1 must be a four-digit unsigned zoned decimal integer item whose picture clause does not contain a "P".
2. Define data-name-1 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
3. Data-name-1 can be qualified.
4. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. Data-name-1 specifies the wait time in seconds for message transmission or reception.
2. The value of data-name-1 must be in the range of 0 to 9999. The value 0 indicates an infinite wait time.
3. The value of data-name-1 is only effective in the following statements and data-name-1 specifies the wait time for message transmission or reception:
   - READ or WRITE statement with a wait request specified in the processing type
4. If the PROCESSING TIME clause is omitted, the message wait time is infinite.

4.3.1.25 RECORD DELIMITER Clause (Sequential File)

This clause specifies the method for determining the length of a variable-length record on an external storage medium.

Format

RECORD DELIMITER IS STANDARD-1

F The COBOL compiler regards the RECORD DELIMITER clause as a comment.

4.3.1.26 RECORD KEY Clause (Indexed File)

This clause specifies the prime record key.

Format

RECORD KEY IS {data-name-1} ...
[WITH DUPLICATES]

Syntax Rules

1. Data-name-1 can be qualified.
2. Data-name-1 must be defined in the record description entry relating to the file-name specified in the RECORD KEY clause.
3. The type of data-name-1 depends on the file system. Refer to the section on using other file systems or comparing different file systems in your "NetCOBOL User's Guide" for details of the types supported by your file system.
4. When the indexed file contains a variable-length record, the first n characters of the variable-length record must contain the prime record key. Here, n represents the size of the smallest record in the file. See the topic titled "RECORD Clause (Sequential file, Relative file and Indexed file)" in "File Description Entry" section of Chapter 5, the "Data Division" for more information.
5. Data-name-1 must not be in a variable position in the record.
6. See Appendix B, "System Quantitative Restrictions," for information on the maximum length of data-name-1 and the maximum number of data-name-1 descriptions allowed.

General Rules

1. The RECORD KEY clause specifies the prime record key of the file.
2. F When the DUPLICATES phrase has been specified, the value of the prime record key need not be unique for each record in the file.

   When the DUPLICATES phrase has been omitted, the value of the prime record key must be unique for each record in the file.
3. The following items must specify identical values as when the file was created:
   a. Data description entry of data-name-1
   b. Relative position in the record of data-name-1
   c. Number of data-name-1 descriptions
   d. F DUPLICATES phrase
4. When associating two or more record description entries with one file, data-name-1 must be specified in only one of the record description entries. The same character position as data-name-1 can be referenced implicitly as a key in any of the other record description entries.

5. When the relevant file connector is an external file connector, all file description entries relating to the file connector in the run unit must conform to the following rules:
    a. Data-name-1 must be in the same relative position in the record for all data description entries.
    b. The contents of the data description entries of data-name-1 must be the same.
    c. The number of data-name-1 descriptions must be the same.
    d. The DUPLICATES phrase must be specified for all data description entries or omitted for all data description entries.
6. When multiple data-name-1 descriptions have been written, they are linked in the sequence in which they were written and treated as a single prime record key.

4.3.1.27 RESERVE Clause (Sequential File, Relative File, Indexed File, and Report Writer)

This clause specifies the number of input-output areas.

Format

RESERVE integer-1 [AREA | AREAS]

The COBOL compiler regards the RESERVE clause as a comment.

4.3.1.28 SELECT Clause (Sequential File, Relative File, Indexed File, and Report Writer)

This clause declares a file-name.

Format

SELECT [OPTIONAL] file-name-1

Syntax Rules

1. The SELECT clause must be specified first in the file description entry.
2. The file description entry of file-name-1 must be included in the program in which the SELECT clause has been specified.

General Rules

1. The OPTIONAL phrase specifies that the physical file represented by file-name-1 does not necessarily exist. A file in which the OPTIONAL phrase has been specified is called an "optional file."
2. The OPTIONAL phrase can be specified for the following files only:
    a. A sequential, relative or indexed file opened in input mode, I-O or extend mode.
    b. Report files opened in the extend mode.

4.3.1.29 SELECT Clause (Sort-merge and Presentation File(*))

This clause declares the name of a presentation file.

*: Extended functions or functions specific to NetCOBOL.

- Presentation File

Format

SELECT file-name-1

Syntax Rules

1. The SELECT clause must be the first clause in the file description entry.
2. The file description entry of file-name-1 must be included in a program in which the SELECT clause has been written.

F 4.3.1.30 SELECTED FUNCTION Clause (Presentation File)

This clause specifies the attention type.

Format

SELECTED FUNCTION IS data-name-1

Syntax Rules

1. Data-name-1 must be a four-character alphanumeric data item or a group item.
   Define data-name-1 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
   - LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)
2. Data-name-1 can be qualified.
3. Data-name-1 must not be located in a variable position in a record.

General Rules

1. Data-name-1 specifies the attention type. Normal execution of a READ statement causes the attention type to be returned from the system and moved to data-name-1.
2. The following values can be set as the attention key in data-name-1.
   a. A key which can be altered when the screen is defined (such as a Function key)
   b. ENTER key
   c. Item select
3. Refer to "NetCOBOL User's Guide" for information on the attention type.

F 4.3.1.31 SESSION CONTROL Clause (Presentation File)

The SESSION CONTROL clause is used to specify or reference session information.

Format

SESSION CONTROL IS data-name-1

Syntax Rules

1. Data-name-1 must be a one-character alphanumeric data item.
   Define the data item in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section

2. Data-name-1 can be qualified.
3. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. Data-name-1 specifies or references session information. When a READ statement has been executed successfully, the system updates data-name-1 with session information. Data-name-1 must be updated before a WRITE statement is executed.
2. Depending on the value of data-name-1, one of the following specifications is provided as session information:
   - Start of conversational processing
   - End of conversational processing
   - Continuation of conversational processing
3. If the SESSION CONTROL clause is omitted, the continuation of conversational processing is assumed when a WRITE statement is executed.

F 4.3.1.32 SYMBOLIC DESTINATION Clause (Presentation File)

This clause specifies the destination type.

Format

SYMBOLIC DESTINATION IS { data-name-1 / literal-1 }

Syntax Rules

1. Data-name-1 must be a three-character alphanumeric data item or a group item.
   Data-name-1 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
   - LOCAL-STORAGE section ([Winx64] ,[Linux64] and [.NET] only)
2. Data-name-1 can be qualified.
3. Data-name-1 must not be located in a variable position in a record.

General Rules

1. Data-name-1 or literal-1 is used for setting and referencing the destination type. Executing an input-output statement causes the destination type in data-name-1 or literal-1 to be updated.
2. The following table lists the values that can be used in data-name-1 and literal_1.

Value in data-name-1 or literal-1	Destination Type
DSP [DS] [HP] [Win16] [Win32] [Winx64] [Solaris] [Linux] [Linux64]	Display unit
PRT	Printing device
ACM [DS] [Win16] [Win32] [Winx64] [Solaris] [Linux64]	Asynchronous message communication
APL	Inter-program communication

Value in data-name-1 or literal-1	Destination Type
[DS] [Win32] [Solaris]	
TRM [DS] [Win32]	Terminal other than display unit and printing device

3. When the SYMBOLIC DESTINATION clause has been omitted, DSP is assumed.

F

4.3.1.33 UNIT CONTROL Clause (Presentation File)

This clause is used to set unit control information.

Format

UNIT CONTROL IS data-name-1

Syntax Rules

1. Data-name-1 must be a six-character alphanumeric data item or a group item.
   Define data-name-1 in one of the following:
   - WORKING-STORAGE section
   - LINKAGE section
   - LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)
2. Data-name-1 can be qualified.
3. Data-name-1 must not be located in a variable position in a record.

General Rules

1. Data-name-1 is used to set special unit control information. Unit control information must be updated prior to executing a WRITE statement.
2. Refer to "NetCOBOL User's Guide" for information on the unit control information.

4.3.2 I-O-Control Paragraph (I-O-CONTROL)

This paragraph specifies the special control technique used by the object program.

Format

I-O-CONTROL.

[[APPLY { MULTICONVERSATION-MODE / MULTICON } clause]

[APPLY SAVED-AREA clause]

[{MULTIPLE FILE TAPE clause} ...]

[{RERUN clause } ...]

[{SAME clause} ...].]

F

The APPLY MULTICONVERSATION-MODE clause is native to [HP], [DS].

The APPLY SAVED-AREA clause is native to [DS].

Syntax Rules

1. The clauses can be written in any order.

2. Files having different organizations can be specified in one clause. However, each clause does not apply to every file organization. The table below shows what clauses apply to which file organizations.

Clause in I-O-control paragraph	Sequential file	Relative file	Indexed file	Sort-merge file	Presentation file	Report file
APPLY MULTICON clause	X	X	X	X	O	X
APPLY SAVED- AREA clause	X	X	X	X	O	X
MULTIPLE FILE TAPE clause	-	X	X	X	X	-
RERUN clause	-	-	-	X	X	X
SAME clause	O	O	O	O	O	O

O : A file of the organization type indicate can be written.

X : A file of the organization type indicate cannot be written.

- : A file of the organization type indicate can be written but it is regarded as a comment.

General Rules

F The COBOL compiler regards the MULTIPLE FILE TAPE clause and RERUN clause as comments.

F 4.3.2.1 APPLY MULTICONVERSATION-MODE Clause (Presentation File)

The APPLY MULTICONVERSATION-MODE clause indicates that multiple conversational processing is performed for two or more destinations.

Format

APPLY { MULTICONVERSATION-MODE / MULTICON }

The APPLY MULTICONVERSATION-MODE clause is native to [HP] and [UXP/DS].

Syntax Rules

1. MULTICONVERSATION-MODE and MULTICON are synonymous.
2. The APPLY MULTICONVERSATION-MODE clause cannot be specified in a nested program.

General Rules

1. The APPLY MULTICONVERSATION-MODE clause specifies that multiple conversational processing can be performed for two or more destinations. If the APPLY MULTICONVERSATION-MODE clause is specified, the system automatically performs the steps of the program execution flow in conversational processing with multiple destinations. Thus, input-output statements can be written in the same way that they are written when processing a single destination.
2. If the APPLY MULTICONVERSATION-MODE clause is specified, the APPLY SAVED-AREA clause must also be specified.
3. The APPLY MULTICONVERSATION-MODE clause is valid for all presentation files defined in the program containing the clause and in its nested programs.

F 4.3.2.2 APPLY SAVED-AREA Clause (Presentation File)

This clause guarantees the value of a data item over several conversations.

Format

APPLY SAVED-AREA TO {data-name-1}...

F

The APPLY SAVED-AREA clause is native to [DS].

Syntax Rules

1. Data-name-1 must be a fixed-length data item of level-number 01 or 77. It must be defined in the WORKING-STORAGE section.
2. A VALUE clause must not be specified in data-name-1 or a data item subordinate to data-name-1.
3. An APPLY SAVED-AREA clause cannot be specified in a nested program.

General Rules

1. Use the APPLY SAVED-AREA clause to guarantee the value of data-name-1 over several conversations.
2. The APPLY SAVED-AREA clause is effective for all presentation files defined in the program in which the clause was written and in all nested programs contained therein.

4.3.2.3 MULTIPLE FILE TAPE Clause (Sequential File and Report Writer)

This clause specifies the position of a file on a multiple file tape. The MULTIPLE FILE TAPE clause is an obsolete language element.

Format

MULTIPLE FILE TAPE CONTAINS

{file-name-1 [POSITION integer-1]}...

F

The COBOL compiler regards the MULTIPLE FILE TAPE clause as a comment.

4.3.2.4 RERUN Clause (Sequential File, Relative File and Indexed File)

This clause specifies the time at which rerun starts. The RERUN clause is an obsolete language element.

Format

RERUN ON file-identifier-1 EVERY

{ [END OF] { REEL / UNIT } / integer-1 RECORDS } OF file-name-1

F

The COBOL compiler regards the RERUN clause as a comment.

4.3.2.5 SAME Clause (Sequential, Relative, Indexed, Sort-merge, Presentation File(*), and Report Writer Files)

This clause specifies that files should share file storage areas.

F

*: Extended functions or functions specific to NetCOBOL.

- Presentation File

Note

The SAME clause cannot be used in [.NET].

Format 1

Sharing files (SAME AREA clause)

SAME AREA FOR file-name-1 {file-name-2} ...

Format 2

Sharing records (SAME RECORD AREA clause)

SAME RECORD AREA FOR file-name-1 {file-name-2} ...

Format 3

Allocating suitable sort-merge file storage areas (SAME SORT/SORT-MERGE AREA clause)

```
SAME    { SORT       }    AREA
        { SORT-MERGE }
        FOR file-name-1 {file-name-2} ...
```

Syntax Rules

1. File-name-1 and file-name-2 must be defined in the FILE-CONTROL paragraph of the same program.
2. The file connectors of file-name-1 and file-name-2 must not be external file connectors.
3. File-name-1 and file-name-2 must be files having one of the following organizations:
   a. In Format 1, they must be a sequential, relative, indexed, presentation, or report file.
   b. In Format 2, they must be a sequential, relative, indexed, presentation, or sort-merge files.
   c. In Format 3, they must be a sort-merge, sequential, relative, or indexed files.
4. File-name-1 and file-name-2 need not have the same organization or same access mode.
5. SORT and SORT-MERGE are synonymous.
6. Two or more SAME clauses can be written in one program as long as they conform to the following rules:
   a. A file-name in a SAME AREA clause must not be repeated in another SAME AREA clause.
   b. A file-name in a SAME RECORD AREA clause must not be repeated in another SAME RECORD AREA clause.
   c. A file-name which has been specified in both the SAME AREA clause and in a SAME RECORD AREA clause, all other file-names which have been specified in the SAME AREA clause must also be specified in the same SAME RECORD AREA clause. However, file-names not specified in a SAME AREA clause can be written in any SAME RECORD AREA clause.
   d. A sort-merge file-name in a SAME SORT/SORT-MERGE AREA clause must not be repeated in another SAME SORT/SORT-MERGE AREA clause.
   e. When a non-sort-merge file has been specified in both the SAME AREA clause and the SAME SORT/SORT-MERGE AREA clause, all other file-names in the SAME AREA clause must also be specified in the same SAME SORT/SORT-MERGE AREA clause.
7. F Files which specify RECORD CONTAINS 0 must not be specified in the SAME AREA or SAME RECORD AREA clause.

General Rules

- Rules for Format 1

  The SAME AREA clause specifies that two or more files share the same storage area during processing. The entire portion of the storage area allocated is shared. Only one of the files specified in the clause can be open at a time.

- Rules for Format 2

  The SAME RECORD AREA clause indicates that the files specified in the clause share the same storage area to process their records. The area which is shared by several records is called a "shared area." The files specified in this clause can be open at the same time. The shared area is used to store records for the files specified. This is equivalent to redefining the shared area for each participating record.

- Rules for Format 3

  The SAME SORT/SORT-MERGE AREA clause indicates that the files specified in the clause will share the same sort storage area, or share the same sort -merge storage area. At least one of the files specified in this clause must be a sort-merge file.

F

The COBOL compiler allocates this storage area, so the SAME SORT/SORT-MERGE AREA clause is regarded as a comment.

Chapter 5 Data Division

The data division defines the data used by an object program. Data that can be defined and processed by object programs includes the following:

- File data entered from external storage and output to external storage or printing device
- File data for sort and merge
- File data for preparing a report
- Data displayed on and entered from a display unit
- Data specified by users

5.1 Composition of the Data Division

The DATA DIVISION follows the ENVIRONMENT DIVISION. The DATA DIVISION is optional and can be omitted.

The DATA DIVISION has eight sections:

F
- BASED-STORAGE SECTION
- FILE SECTION
- WORKING-STORAGE SECTION
- LOCAL-STORAGE SECTION

F
- CONSTANT SECTION
- LINKAGE SECTION
- REPORT SECTION

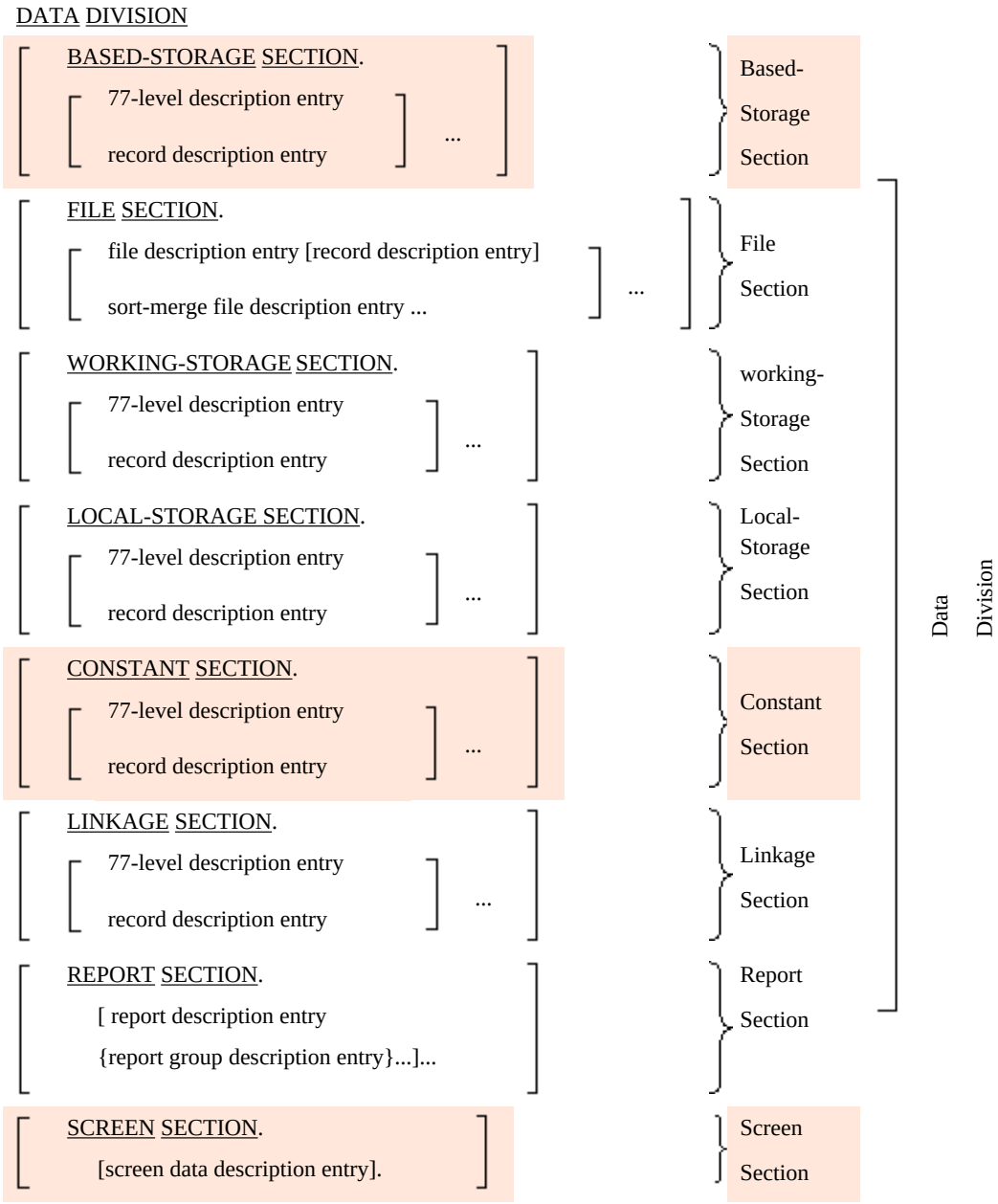
F
- SCREEN SECTION

Note

The report writer module and the screen handling module cannot be used in [LinuxIPF] and [.NET].
The report writer module cannot be used in [Winx64].
The screen handling module cannot be used in [Linux], [Linux64].

The DATA DIVISION configuration is shown below. Clauses and entries in the DATA DIVISION must be written in the following order.

Format



F

BASED-STORAGE SECTION

The BASED-STORAGE section is used to describe data in a secure area. Unlike the WORKING-STORAGE section, data defined in the BASED-STORAGE section occupy no memory until referenced with a pointer data item. One use of the BASED-STORAGE section would be to describe a data item, allocate memory for the data item, save the address of the allocated memory in a pointer data item, and when associated with the data item in BASED-STORAGE, the data item can now be manipulated. See the topic "Pointer Handling" in the "Nucleus" section of Chapter 2, "COBOL Modules" for an example of using pointers and the BASED-STORAGE section.

The section header of the BASED-STORAGE SECTION is followed by 77-level description entries and/or record description entries.

FILE SECTION

The FILE SECTION defines the file structure of sequential, relative, indexed, sort-merge, report, and presentation files, as well as the structure of the file's records.

Files defined in the FILE SECTION must be associated with external media via a file control entry in the ENVIRONMENT DIVISION. The section header of the FILE SECTION can contain the following entries.

- When a sequential , relative , indexed , or presentation file is used, one file description entry is written for each file, followed by one or more record description entries. The record description entries must directly follow the file description entry.
- When a sort-merge module is used, one sort-merge file description entry is written for each sort-merge file, followed by one or more record description entries. The record description entries must directly follow the sort-merge file description entry.
- When a report writer module is used, one file description entry is written for each report file. The FILE SECTION does not define the data structure of report file records, rather, report file records are defined in the REPORT SECTION of the DATA DIVISION.

WORKING-STORAGE SECTION

The WORKING-STORAGE SECTION defines the structure of data and records used in a COBOL program.

The section header of the WORKING-STORAGE section is followed by 77-level description entries and/or record description entries.

LOCAL-STORAGE SECTION

Data items defined in the local-storage section are automatic items.
The storage of local-storage section is allocated and set to initial state each time when the program is activated.

Each activated instance of the program has its own copy of the item that persists while that instance of the program is in active state.

The section header of the LOCAL-STORAGE SECTION is followed by 77-level description entries and/or record description entries.

The LOCAL-STORAGE SECTION is available for [Winx64], [Linux64] and [.NET].

F

The recursive attribute can be assigned to a program by specifying the LOCAL-STORAGE SECTION.

F

CONSTANT SECTION

The CONSTANT SECTION defines constants used by a COBOL program. Values specified in the CONSTANT SECTION do not change during program execution.

The section header of the CONSTANT section is followed by 77-level description entries and/or record description entries.

LINKAGE SECTION

The LINKAGE SECTION describes data passed from a calling program to a called program.

The LINKAGE SECTION is written in called programs only when the calling program specifies the USING phrase on the CALL statement. The called program must describe the data being passed to it exactly as it was defined in the calling program, and the LINKAGE SECTION must contain the same number of data items as is used in the CALL...USING statement. The order of the data items in the LINKAGE SECTION need not correspond to the order of the data items specified in the CALL...USING statement.

The called program's PROCEDURE DIVISION or ENTRY statement(*) must also specify a USING phrase. The called program's PROCEDURE DIVISION or ENTRY statement USING phrase must describe the data being passed in the same order as the calling program's CALL...USING statement. All data items described in the called program's PROCEDURE DIVISION or ENTRY statement USING clause must be described in the called program's LINKAGE SECTION.

The section header of the LINKAGE SECTION is followed by 77-level description entries and/or record description entries.

F

*: Extended functions or functions specific to NetCOBOL.

- ENTRY statement

REPORT SECTION

The REPORT SECTION defines reports for the report writer module. Following the REPORT SECTION header is the description of one or more reports. The report defined in the REPORT SECTION must be associated with a report file in the FILE SECTION of the DATA DIVISION.

The section header of the REPORT section is followed by report description entries or report group description entries. One report description entry is written for each report, followed by one or more report group description entries. The report group description entries must directly follow the report description entry.

Note

The following clauses cannot be specified on a report group description entry.

- CHARACTER TYPE clause
- PRINTING POSITION clause
- ENCODING clause

The NetCOBOL compiler does not issue diagnostic messages when these clauses are specified on a report group description entry.

F

SCREEN SECTION

The SCREEN SECTION defines character based screen structures, and the data displayed and/or data entered from the screen. When the screen handling module is used, the SCREEN section must be written.

The section header of the SCREEN section is followed by one or more screen data description entries.

77-level Description Entry

Data items that are not interdependent and that cannot be classified further can be defined as 77-level description entries. All of the following clauses must be written in a 77-level description entry:

- Level-number 77
- Data-name
- The PICTURE clause, and USAGE IS INDEX, USAGE IS COMP-1, USAGE IS COMP-2, or USAGE IS POINTER clauses.

Record Description Entry

Data which is logically interdependent can be grouped together and defined in a "record description entry`. A record description entry can define one or more group and/or elementary items.

Record description entries are described using level numbers, data item names, and PICTURE clauses. Level numbers are COBOL concepts that denote the hierarchy of data items in a record. Level number 01 denotes the highest level in a record description entry and can also be used to name a record. Level numbers 02 to 49 are all subordinate to a level 01 item, and must be used following a level 01 entry. Level numbers 02 to 49 are used to describe the data items that, taken as a whole, comprise the record. Level numbers of data items lower than those of level-number 01 need not be consecutive.

Data items are described as "group items" or "elementary items". A group item is a data item that is comprised of one or more group items and/or elementary items. That is, a group item can be subdivided into one or more data items. An elementary item is a data item that cannot be subdivided further. A group item is composed of all the data items following whose level numbers are less than the group item's level number. An elementary item is a data item whose level number is equal to or less than both the preceding and following data item's level number.

An example of level numbers follows:

```
01  ITEM-1.                                  *>...1
     05  ITEM-2  PIC X(4).                   *>...2
     05  ITEM-3.                             *>...3
           10  ITEM-4   PIC X(10).           *>...4
```

```
           10  ITEM-5   PIC 9(9).                *>...5
     05  ITEM-6         PIC 9(9) COMP-5.         *>...6
01  ITEM-7   PIC +9(4).99.                       *>...7
```

1. A group item named ITEM-1 is defined.
2. ITEM-2 is an elementary item belonging to group item (record) ITEM-1.
3. ITEM-3 is a group item belonging to group item ITEM-1.
4. ITEM-4 is an elementary item belonging to group item ITEM-3.
5. ITEM-5 is an elementary item belonging to group item ITEM-4.
6. ITEM-6 is an elementary item belonging to group item ITEM-1.
7. ITEM-7 is a group item, a record, and an elementary item.

5.2 File Description Entry

A file description entry defines the physical structure of a file. A file description entry is written in the FILE SECTION or REPORT SECTION of a program.

Format 1

Sequential file

FD file-name-1
[BLOCK CONTAINS clause]
[CODE-SET clause]
[CONTROL RECORDS clause]
[DATA RECORDS clause]
[ENCODING clause]
[EXTERNAL clause]
[GLOBAL clause]
[LABEL RECORDS clause]
[LINAGE clause]
[RECORD clause]
[VALUE OF clause].

Format 2

Relative file/indexed file

FD file-name-1
[BLOCK CONTAINS clause]
[DATA RECORDS clause]
[ENCODING clause]
[EXTERNAL clause]
[GLOBAL clause]
[LABEL RECORDS clause]
[RECORD clause]
[VALUE OF clause] .

F

Format 3

Presentation file

```
FD file-name-1
    [ENCODING clause]
    [EXTERNAL clause]
    [GLOBAL clause]
    [RECORD clause] .
```

Format 4

Report file (file used in report writer module)

```
FD file-name-1
    [BLOCK CONTAINS clause]
    [CODE-SET clause]
    [EXTERNAL clause]
    [GLOBAL clause]
    [LABEL RECORDS clause]
    [RECORD clause]
    [REPORT clause]
    [VALUE OF clause] .
```

Syntax Rules

1. Level indicator FD must be written at the beginning of a file description entry, followed by file-name-1. The order of clauses that follow file-name-1 is optional.
2. For a non-report file, the file description entry must be followed by one or more record description entries. For a report file, the report description entry is described in the REPORT SECTION.
3. For a report file, file-name-1 must be a sequential file.

General Rules

1. A file description entry relates file-name-1 to a file connector.
2. The DATA RECORDS, LABEL RECORDS, and VALUE OF clauses are obsolete language elements.

F

3. This compiler regards the BLOCK CONTAINS, DATA RECORDS, LABEL RECORDS, and VALUE OF clauses as comments.

5.2.1 BLOCK CONTAINS Clause (Sequential File, Relative File, Indexed File, and Report Writer Module)

The BLOCK CONTAINS clause specifies the block size.

Format

```
[BLOCK CONTAINS
    [integer-1 TO] integer-2  { RECORDS    }
                              { CHARACTERS }
```

General Rule

F

This compiler regards the BLOCK CONTAINS clause as a comment.

5.2.2 CODE-SET Clause (Sequential File and Report Writer Module)

The CODE-SET clause specifies the character codes used by the file.

Format

CODE-SET IS code-name-1

Syntax Rules

1. The CODE-SET clause can be written only when both of the following conditions are satisfied:
   - All data items included in a record description entry for the files are display data items.
   - The SIGN IS SEPARATE clause is specified on all signed numeric data items.
2. In [Win32], [Winx64] and [Linux64], code-name-1 must not be associated with any of the following in the ALPHABET clause in the SPECIAL-NAMES paragraph.
   - literal
   - SJIS
   - UTF8
   - UTF16
   - UTF16LE
   - UTF16BE
   - UTF32
   - UTF32LE
   - UTF32BE

   In other operating systems, code-name-1 must not be associated with a literal in the ALPHABET clause in the SPECIAL-NAMES paragraph.
3. The CODE-SET clause cannot be specified for a print file.

General Rule

F

This compiler regards the CODE-SET clause as a comment.

F

5.2.3 CONTROL RECORDS Clause (Sequential File)

The CONTROL RECORDS clause specifies the name of a control record in a file.

Format

CONTROL { RECORD IS / RECORDS ARE } {data-name-1} ...

Syntax Rules

1. Data-name-1 must be a data-name of level-number 01 in a record description entry associated with the file.
2. Data-name-1 must be unique in a record description entry associated with the file.

3. Data-name-1 can be qualified.
4. The CONTROL RECORDS clause can be specified only for a print file with the FORMAT clause.

General Rules

1. When the record specified by data-name-1 is output with the WRITE statement, it is handled as a control record. A control record posts the record editing method to the system.
2. A control record associated with a file shares the record area with other records associated with the file.

5.2.4 DATA RECORDS Clause (Sequential File, Relative File, and Indexed File)

The DATA RECORDS clause lists names of the data records that are contained in the file.

The DATA RECORDS clause is an obsolete language element.

Format

DATA { RECORD IS / RECORDS ARE } {data-name-1} ...

Syntax Rule

Data-name-1 must be a data-name of level-number 01 in a record description entry related to the file.

General Rule

F

This compiler regards the DATA RECORDS clause as a comment.

F

5.2.5 ENCODING Clause (Sequential File, Relative File, Indexed File, and Presentation File)

The ENCODING clause specifies the encoding form of the file.

Format

ENCODING IS alphabet-name-1 [OR alphabet-name-2]

The ENCODING clause is specific to [Win32], [Winx64] and [Linux64].

Syntax Rules

1. Alphabet-name-1 and alphabet-name-2 must be an alphabet-name associated with any of the following in the ALPHABET clause.
   - SJIS
   - UTF8
   - UTF16
   - UTF16LE
   - UTF16BE
   - UTF32
   - UTF32LE
   - UTF32BE

2. When specifying alphabet-name-2, it must not be same as alphabet-name-1 in the ALPHABET clause FOR phrase.

General Rule

1. When ALPHANUMERIC is specified either explicitly or implicitly in the alphabet-name, then the encoding form is applicable in the alphanumeric data item or alphanumeric edited data item.
2. When NATIONAL is specified in alphabet-name, then the encoding form is applied to the national item or national edit items.

5.2.6 EXTERNAL Clause (Sequential File, Relative File, Indexed File, Presentation File(*), and Report Writer Module)

The EXTERNAL clause gives an external attribute to a file connector.

F

* : Extended functions or functions specific to NetCOBOL.

- Presentation File

Format

IS EXTERNAL

General Rule

The EXTERNAL clause specifies that the file connector is an external file connector. External file connectors that are defined in other COBOL modules within the run unit in exactly the same manner and with the same name share the external file connector's record area and have equal access to the data file.

5.2.7 GLOBAL Clause (Sequential File, Relative File, Indexed File, Presentation File(*), and Report Writer Module)

The GLOBAL clause specifies that file-name-1 is a global name.

F

* : Extended functions or functions specific to NetCOBOL.

- Presentation File

Format

IS GLOBAL

Syntax Rule

The GLOBAL statement must not be written in a file description entry associated with a file that specifies the SAME RECORD AREA clause.

General Rule

The GLOBAL clause specifies that the related file-name-1 is a global name. A global name can be accessed by the program declaring the global name, or from any program that is contained within the program declaring the global name.

5.2.8 LABEL RECORDS Clause (Sequential File, Relative File, Indexed File, and Report Writer File)

The LABEL RECORDS clause specifies existence or non-existence of a file label. The LABEL RECORDS clause is an obsolete language element.

Format

```
LABEL   { RECORD IS  }  { STANDARD }
        { RECORDS ARE}  { OMITTED  }
```

General Rule

F

This compiler regards the LABEL RECORDS clause as a comment.

5.2.9 LINAGE Clause (Sequential File)

The LINAGE clause defines the configuration of logical pages.

Format

```
LINAGE IS { data-name-1 } LINES
          { integer-1   }
    [ WITH FOOTING AT { data-name-2 } ]
                      { integer-2   }
    [ LINES AT TOP    { data-name-3 } ]
                      { integer-3   }
    [ LINES AT BOTTOM { data-name-4 } ]
                      { integer-4   }
```

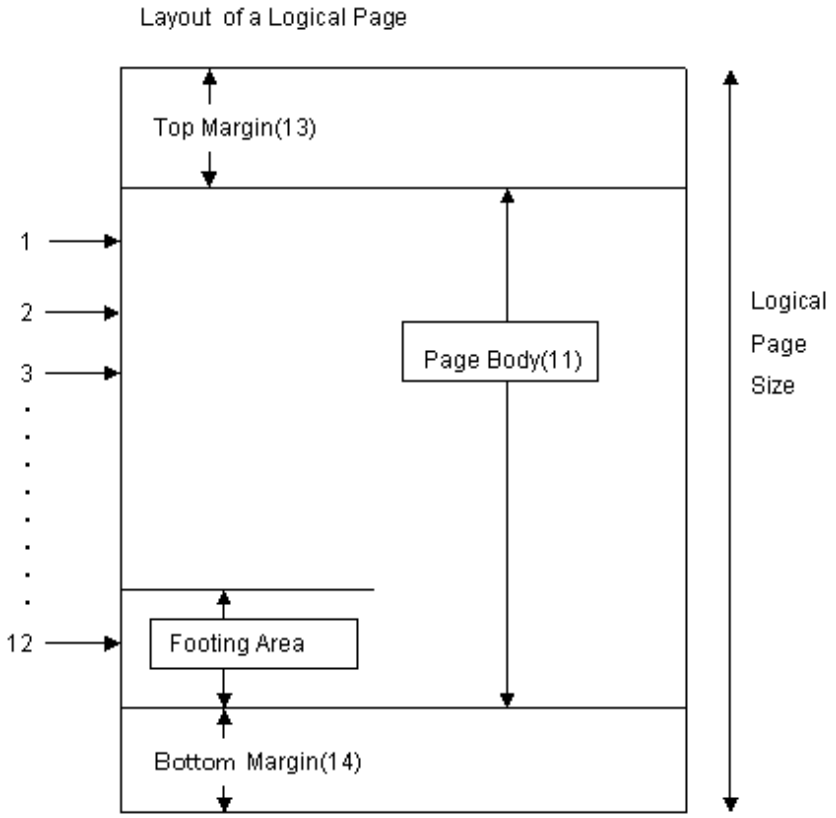
Syntax Rules

1. Data-name-1 to data-name-4 must be unsigned integer items.
2. Data-name-1 to data-name-4 may be qualified.
3. Integer-2 must be equal to or less than integer-1.
4. Zero can be specified in integer-3 and integer-4.
5. The LINAGE clause can be specified only in print files not specifying the FORMAT clause.
6. For the maximum number of lines to make up a logical page, see Appendix B, "System Quantitative Restrictions."

General Rules

1. Specification of the LINAGE clause and the corresponding logical page configuration are indicated below.

```
LINAGE IS I1 LINES
    WITH FOOTING AT I2
    LINES AT TOP I3
    LINES AT BOTTOM I4
```



The I1 line starting from the line below the top margin is called the "page body."

The I2 line from the first line of the page body to the end line of the page body is called "footing area."

2. The size of a logical page is the total of the values written in each phrase other than the FOOTING phrase. If the LINES AT TOP phrase is omitted, "LINES AT TOP 0" is assumed. In the same way, if the LINES AT BOTTOM phrase is omitted, "LINES AT BOTTOM 0" is assumed.
3. If the FOOTING phrase is omitted, the end-of-page condition does not occur independently of the page overflow condition. There is no need to specify the size of a logical page according to the size of a physical page.
4. Integer-1 or data-name-1 specifies the number of lines that compose a logical page. This value must be positive.
5. Integer-2 or data-name-2 specifies the line position to start the footing area on the page body. The first line of the page body is 1. The value must be positive and equal to or less than integer-1 or data-name-1.
6. Integer-3 or data-name-3 specifies the number of lines of the top margin of a logical page. Zero can be specified.
7. Integer-4 or data-name-4 specifies the number of lines of the bottom margin of a logical page. Zero can be specified.
8. If integer-1, integer-3, and integer-4 are specified, the number of lines that compose each part of a logical page is determined according to the values when the OPEN OUTPUT statement is executed. If integer-2 is specified, the footing area is determined according to the value when the OPEN OUTPUT statement is executed. These integer values are applied to all logical pages output to the file.
9. If data-name-1, data-name-3, and data-name-4 are specified, the number of lines that compose each part of a logical page is determined as follows:
    a. When the OPEN OUTPUT statement is executed, the number of lines that compose each part of the first logical page is determined according to these data items.
    b. When the WRITE statement with the ADVANCING PAGE phrase is executed, or the page overflow condition occurs, the number of lines that compose each part of the next logical page is determined according to these data items.
10. If data-name-2 is specified, the number of lines that compose the footing area of the first logical page is determined according to the value of data-name-2 when the OPEN OUTPUT statement is executed. When the WRITE statement

with the ADVANCING PAGE phrase is executed, or the page overflow condition occurs, the footing area of the next logical page is determined according to the value of this data item.

11. If the LINAGE clause is specified, the compiler automatically generates a special register named LINAGE-COUNTER. A value indicating the current line position of the page body of the logical page is always contained in LINAGE-COUNTER. Rules for the LINAGE-COUNTER are listed below.

    a. One LINAGE-COUNTER is generated for each file specifying the LINAGE clause.

    b. Statements in the PROCEDURE DIVISION may reference the value of LINAGE-COUNTER, but LINAGE-COUNTER cannot be initialized by any PROCEDURE DIVISION statements. If two or more files are defined and specify the LINAGE clause, the LINAGE-COUNTER must be qualified by a file-name.

    c. THE VALUE OF the LINAGE-COUNTER automatically changes according to the following rules during the execution of the WRITE statement.

       - If the WRITE statement with the ADVANCING PAGE phase is executed, the value of the LINAGE-COUNTER is set to 1.
       - If the WRITE statement's ADVANCING phrase specifies an integer or identifier, when executed, the value of the LINAGE-COUNTER is incremented by the value of the integer or identifier specified in the ADVANCING phrase.
       - If the WRITE statement without the ADVANCING phrase is executed, the value of the LINAGE-COUNTER is incremented by one.
       - The value of the LINAGE-COUNTER is automatically reset to one when the device is repositioned to the first line that can be written on for each of the succeeding logical pages.

    d. The OPEN OUTPUT statement initializes the LINKAGE-COUNTER to the value 1.

12. If a file connector associated with the file which specified the LINAGE clause is an external file connector, all file description entries associated with the file connector in the run unit must adhere the following rules:

    a. The LINAGE clause must be written in all file description entries associated with the file connector.

    b. When integer-1 to integer-4 are written, the same values must be specified in all file description entries associated with the file connector.

    c. When data-name-1 to data-name-4 are specified, these data-names must be external data items. Further, all of the corresponding data-names in file description entries associated with the file connector must be identical.

5.2.10 RECORD Clause (Sequential File, Relative File, and Indexed File)

The RECORD clause specifies the size and format of records in a file.

Format 1

Fixed length records.

RECORD CONTAINS integer-1 CHARACTERS

Format 2

Variable-length records.

RECORD IS VARYING IN SIZE

[[FROM integer-2] [TO integer-3] CHARACTERS]

[DEPENDING ON data-name-1]

Format 3

Record description entry.

RECORD CONTAINS

integer-4 TO integer-5 CHARACTERS

Syntax Rules

- Rules for Format 1

  No record description entry for the file can contain more than integer-1 characters.

- Rules for Format 2

  1. A record having fewer than integer-2 or more than integer-3 character positions must not be defined in the record description entry associated with the file.
  2. Integer-3 must be greater than integer-2.
  3. Data-name-1 must be an unsigned integer item defined in one of the following:
     - WORKING-STORAGE SECTION
     - LINKAGE SECTION
     - LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)
  4. The DEPENDING ON phrase cannot be specified in any of the following conditions:
     a. When the CHARACTER TYPE or PRINTING POSITION clause is specified in the record description entry associated with a file
     b. When the record associated with a file is output with a WRITE statement specifying the FROM phrase and one of the following data items is specified in the FROM phrase:

     - A data item which specified the CHARACTER TYPE or the PRINTING POSITION clause

     - A data item subordinate to a data item which specifies the CHARACTER TYPE or the PRINTING POSITION clause

     - A group data item that contains a data item which specifies the CHARACTER TYPE or the PRINTING POSITION clause.

- Rules for Format 3

  Integer-5 must be greater than integer-4.

General Rules

- Rules common to format 3 and when the RECORD clause is omitted.

  1. If a format 3 RECORD clause is specified, or the RECORD clause is omitted, the size of a data record is determined according to the description of the record description entry.
  2. If a file connector associated with a file is an external file connector, all record description entries associated with the file connector in the run unit must be of the same length.

- Rules for Format 1

  1. The RECORD clause of format 1 is used to define the size of fixed length records. integer-1 specifies the number of characters in each record in the file.

     However, if 0 was specified in the integer-1, the size of the data record is decided depending on the description in the record description entry.

2. The size of a record excludes any characters inserted by the COBOL Run Time System (such as carriage return and line feed characters).

3. If the file connector associated with this file is an external file connector, the same value must be specified for integer-1 in all file description entries associated with the file connector in the run unit.

- Rules for Format 2

1. The RECORD clause of format 2 is used to define the size of variable-length records. Integer-2 specifies the minimum number of characters and integer-3 specifies the maximum number of characters in a record for the file.

F

However, if the number of character positions is specified by the system at runtime, this value is the maximum number of character positions.

2. The minimum and maximum record sizes exclude any control characters inserted by the COBOL Run Time System (such as carriage return and line feed characters).

3. If a file connector associated with a file is an external file connector, the same values must be specified in integer-2 and integer-3 in all file description entries associated with the file connector in the run unit.

F

4. If the record size specified by the RECORD clause is greater than the number of character positions of an associated record description entry, when the record is output, the following values are added to the end of the record:

   a. If the CHARACTER TYPE or the PRINTING POSITION clause is specified in the record description entry, the record is padded with spaces.

   b. If neither the CHARACTER TYPE nor the PRINTING POSITION clause is specified in the record description entry, the excess portion of the record is undefined.

5. "Record size" is the sum of the number of character positions required to store all fixed length elementary items (excluding items that specify the REDEFINES or RENAMES clause) plus the sum of the maximum number of characters in any variable length item subordinate to the record plus the number of character positions added by synchronization. If the OCCURS clause with the DEPENDING ON phrase is specified in a record description entry, the minimum and maximum sizes of the record description entry are determined according to the following rules:

   a. The minimum size of a record description entry is determined by using the minimum number of table elements.

   b. The maximum size of a record description entry is determined by using the maximum number of table elements.

6. If the FROM phrase is omitted, it is assumed that the minimum size of a record description entry is integer-2.

7. If the TO phrase is omitted, it is assumed that the maximum size of a record description entry is integer-3.

8. If the DEPENDING ON phrase is written, the value of data-name-1 is used as follows:

   a. The number of characters of the record to be output must be specified in data-name-1 before the RELEASE, REWRITE, or WRITE statement is executed. When these statements are executed, the number of characters of the record to be output is determined by the value of data-name-1.

   b. If the execution of a READ or RETURN statement is successful, the number of character of the record read is moved to data-name-1.

   c. If the INTO phrase is specified in a READ statement, the value in data-name-1 is used as the number of character of the sending data item of the implicit MOVE statement.

   d. The DELETE, RELEASE, REWRITE, START and WRITE statement do not update data-name-1, it remains unchanged. Data-name-1 is also not changed if execution of a READ statement is unsuccessful.

9. If the DEPENDING ON phrase is omitted, the number of character position of a record output is determined according to the following rules when a RELEASE, REWRITE, or WRITE statement is executed:

    a. If the record does not include variable occurrence data items, the number of characters output is the number of character positions in the record.

    b. If the record includes variable occurrence data items, the number of characters output is the sum of the total of the size of the table (calculated by the value of the data-name of the DEPENDING ON phrase of the OCCURS clause) and the size of the data items other than variable occurrence data items.

10. If the DEPENDING ON phrase is omitted and the INTO phrase is specified in a READ statement, the value of data-name-1 when the DEPENDING ON phase is written is used as the number of character positions of the sending data item of the implicit MOVE statement.

F

5.2.11 RECORD Clause (Presentation File)

The RECORD clause specifies the size and format of records.

Format 1

Variable length records.

RECORD IS VARYING IN SIZE
    [[FROM integer-2] [TO integer-3] CHARACTERS]
    [DEPENDING ON data-name-1]

Format 2

Variable length records.

RECORD CONTAINS
    integer-4 TO integer-5 CHARACTERS

Format 1 and format 2 are the same as format 2 and format 3 of the RECORD clause of sequential files, respectively. See the topic titled "RECORD clause (Sequential File, Relative File, and Indexed File)" above.

5.2.12 RECORD Clause (Report Writer Module)

The RECORD clause specifies the size and format of records.

Format 1

Fixed length records.

RECORD CONTAINS integer-1 CHARACTERS

Format 2

Variable length records.

RECORD CONTAINS
    integer-4 TO integer-5 CHARACTERS

Format 1 and format 2 are the same as format 1 and format 3 of the RECORD clause of sequential files, respectively. See the topic titled "RECORD clause (Sequential File, Relative File, and Indexed File)" above.

5.2.13 REPORT Clause (Report Writer Module)

The REPORT clause relates report-names to report files.

Format

```
{ REPORT IS  }
{ REPORTS ARE } {report-name-1} ...
```

Syntax Rules

1. Report-name-1 must be a report-name written in a report description entry of the REPORT SECTION. If more than one report-name-1 is specified, they may be specified in any order.
2. Only one report-name per report description entry may be specified in a single REPORT clause.
3. The file that specified the REPORT clause is called a report file. A report file may only be specified in the USE, CLOSE, OPEN OUTPUT, and the OPEN EXTEND statements.

General Rules

1. If two or more reports are associated with one report file, two or more report-name-1s are written.
2. During the time between execution of the INITIATE and the TERMINATE statements for one report file, an input-output statement which references the report file must not be executed.
3. If a file connector associated with the report file is an external file connector, the same file description entries must be specified all other programs that are associated with the external file connector.

5.2.14 VALUE OF Clause (Sequential File, Relative File, Indexed File, and Report Writer Module)

The VALUE OF clause defines the value of data items in label records. The VALUE OF clause is an obsolete language element.

Format

```
VALUE OF { data-name-1 IS { data-name-2 } } ...
         {                { literal-1   } }
```

Syntax Rules

1. Data-name-2 may be qualified. Data-name-2 must not be an index data item.
2. Data-name-2 must be a data item defined in the WORKING-STORAGE SECTION.

General Rule

F

This compiler regards the VALUE OF clause as a comment.

5.3 Sort-Merge File Description Entry

The sort-merge file description entry defines the physical structure of sort-merge files. It is specified in the FILE SECTION.

Format

```
SD file-name-1
    [DATA RECORDS clause]
```

[ENCODING clause]

[RECORD clause].

The ENCODING clause is specific to [Win32], [Winx64] and [Linux64].

Syntax Rules

1. Level indicator SD must be written at the beginning of a sort-merge file description entry, followed by file-name-1. The order of the clauses that follow file-name-1 are optional.
2. A sort-merge file description entry must be followed by at least one record description entry. However, an input-output statement must not be executed on a sort-merge file.

General Rules

Rules for the DATA RECORDS, RECORDS and ENCODING clauses of sort-merge files are the same as those of sequential files. See the topic titled "Data Records Clause (Sequential File, Relative File, and Indexed File)" and "ENCODING Clause (Sequential File, Relative File, Indexed File, and Presentation File)" earlier in this section.

5.4 Data Description Entry

A data description entry defines data items. Data description entries may be written in the following:

F - BASED-STORAGE SECTION

- FILE SECTION
- WORKING-STORAGE SECTION
- LOCAL-STORAGE SECTION ([Winx64], [Linux64] and [.NET] only)

F - CONSTANT SECTION

- LINKAGE SECTION

Data description entries are used to define record description and 77-level description entries. A data description entry has three formats: one for defining data-names, one for renaming data-names, and one for defining condition-names.

Format 1

Defining data-names.

level-number [data-name-1 | FILLER]

[BASED ON clause]
[BLANK WHEN ZERO clause]
[CHARACTER TYPE clause]
[ENCODING clause]
[EXTERNAL clause]
[GLOBAL clause]
[JUSTIFIED clause]
[OCCURS clause]
[PICTURE clause]
[PRINTING POSITION clause]
[REDEFINES clause]
[SIGN clause]

[SYNCHRONIZED clause]

[TYPE clause]

[TYPEDEF clause]

[USAGE clause]

[VALUE clause].

The ENCODING clause is specific to [Win32], [Winx64] and [Linux64].

Format 2

Renames data-names.

66 data-name-1

RENAMES clause.

Format 3

Defines condition names.

88 condition-name-1

VALUE clause.

Syntax Rules

- Rules for Format 1

1. Level-number must be between 01 and 49 or 77.
2. Level-number must be specified at the beginning of a data description entry. If a data-name or FILLER is to be used, it must follow the level-number. If the REDEFINES clause is specified, it must follow data-name or FILLER (if present). If data-name or FILLER is not specified, the REDEFINES clause must immediately follow level-number.

   F

   If specifying a TYPEDEF clause, data-name must be specified first, then the keyword TYPEDEF.

   Other clauses are allowed in any arbitrary order.
3. The PICTURE clause is specified when defining elementary items except for the items as follows:
   - index data items
   - BINARY-CHAR/SHORT/LONG/DOUBLE data items

   F

   - internal floating-point data items
   - pointer data items
4. When defining group items, the BLANK WHEN ZERO, JUSTIFIED, PICTURE, and SYNCHRONIZED clauses must not be specified. In [Win32], [Winx64], [Solaris],[LinuxIPF], [Linux64] and [.NET], the SYNCHRONIZED clause can also be specified for a group item.
5. The EXTERNAL clause may only be specified in data description entries of the WORKING-STORAGE SECTION whose level-numbers are 01.
6. The EXTERNAL and REDEFINES clauses may not be specified in the same data description entry.
7. The GLOBAL clause may only be specified in the data description entries of data items having a level-number of 01.
8. If the GLOBAL or EXTERNAL clauses are specified, data-name-1 must be specified.
9. If a file description entry specified the EXTERNAL or GLOBAL clauses, the associated record description entry must specify data-name-1.

F

10. The TYPEDEF clause may only be specified in data description entries containing level-numbers of 01. The TYPEDEF clause may not be used with the REDEFINES or EXTERNAL phrases.
11. The TYPE clause may only be used with the EXTERNAL, GLOBAL, OCCURS, BASED ON, TYPEDEF and VALUE clauses.

- Rules for Format 2

  Level-number 66 must be specified first in a data description entry, followed by data-name-1, then the RENAMES clause.

- Rules for Format 3

  Level-number 88 must be specified first in a data description entry followed by condition-name-1, then the VALUE clause.

General

- Rules for Format 1

  1. Data-name-1 specifies the name of a data item. FILLER is used for defining data items that are not explicitly referenced.
  2. If data-name-1 and FILLER are omitted, FILLER is assumed.
  3. An item defined using the FILLER clause in a data description entry is called a "FILLER data item." A FILLER data item cannot be directly referenced.
  4. If two or more data description entries contain a level-number 01 and are subordinate to level indicators of FD or SD, then the level-number 01`s represent implicit redefinition of the same area.

- Rules for Format 3

  1. A data description entry of format 3 is called "condition-name data description entry." Condition-name-1 specifies the name of a condition, and the VALUE clause specifies a value or range of values of condition-name-1.
  2. A condition-name data description entry must directly follow a data description entry of a conditional variable. The conditional variable must not be any of the following:

     a. A condition-name
     b. A data item of level-number 66
     c. Group items that have a USAGE clause specifying other than the JUSTIFIED, SYNCHRONIZED, or USAGE IS DISPLAY clauses
     d. An INDEX data item

F

     e. A data description item referring to a type that is defined as a group item or that includes the condition-name

F

Notes on Data Description Entries in the BASED-STORAGE SECTION

1. A data description entry in the BASED-STORAGE SECTION must not contain the VALUE clause unless the data item is a condition-name data description entry (level-number 88). Initial values of data items defined in the BASED-STORAGE SECTION are not permitted.
2. The EXTERNAL, CHARACTER TYPE, and PRINTING POSITION clauses may not be specified in data description entries in the BASED-STORAGE SECTION.
3. Data items defined in the BASED-STORAGE SECTION can be written in place of "identifier" in all syntax diagrams.
4. Data items defined in the BASED-STORAGE SECTION can be written in place of "data-name" in the following locations. However, these data-names must not use explicit pointers.

   - Data-name of the REDEFINES clause
   - Data-name of the RENAMES clause
   - Data-name of the THROUGH phrase of the RENAMES clause

- Data-name of the KEY phrase of the OCCURS clause
- Data-name of the WHEN phrase of the SEARCH ALL statement

5. If the data item defined in the BASED-STORAGE SECTION is specified in the data-name of the WHEN phrase of the SEARCH ALL statement, the identifier of the SEARCH ALL statement must use an implicit pointer.
6. A STRONG type declaration may not be referred to in the BASED-STORAGE SECTION.

Notes on Record Description Entries in the FILE SECTION

1. Data description entries in the FILE SECTION, other than condition names entries, may not specify the VALUE clause. Initial values of data items defined in the FILE SECTION are undefined.
2. In the FILE SECTION, the ENCODING clause cannot be specified in the record of the following files.
   - Print File
   - Presentation file where display or printer is specified in the destination type.
3. When the ENCODING clause is not specified in the data items defined in FILE SECTION, then the ENCODING clause specified in the file description is applied to the data items.

Notes on Data Description Entries in the WORKING-STORAGE SECTION

Initial values can be specified in data items other than index data items defined in the WORKING-STORAGE SECTION. The VALUE clause is written to specify initial values. If the VALUE clause is omitted, initial values of data items are undefined.

Notes on Data Description Entries in the LOCAL-STORAGE SECTION

1. Local storage memory is allocated only while the subroutine is active.
2. Data description entries in the LOCAL-STORAGE SECTION may not use an EXTERNAL clause and GLOBAL clause.

The LOCAL-STORAGE SECTION is available in [Winx64], [Linux64] and [.NET].

F

Notes on Data Description Entries in the CONSTANT SECTION

CONSTANT SECTION data items must be initialized by using the VALUE clause.

Data items defined in the CONSTANT SECTION may be specified anywhere data items in the WORKING-STORAGE SECTION may be used.

Notes on Data Description Entries in the LINKAGE SECTION

Data description entries in the LINKAGE SECTION, other than condition-name entries, may not use a VALUE clause. The initial values of data items defined in the LINKAGE SECTION are undefined.

F

5.4.1 BASED ON Clause

An implicit pointer qualifier is specified in a data item of the BASED-STORAGE SECTION.

For details about how to use the examples, refer to "COBOL Syntax Samples", "BASED ON Clause".

Format

BASED ON data-name-1

Syntax Rules

1. The BASED ON clause can only be specified in a data description entry of level-numbers 01 or 77 in the BASED-STORAGE SECTION.
2. The REDEFINES clause must not be specified in a data description entry that contains the BASED ON clause.

3. A TYPE clause that references a strongly-typed item must not be specified in a data description entry that contains the BASED ON clause.
4. Data-name-1 must be a pointer data item defined in the following:
   - WORKING-STORAGE SECTION
   - LINKAGE SECTION
   - LOCAL-STORAGE SECTION ([Winx64], [Linux64] and [.NET] only)
5. Data-name-1 can be qualified.

General Rules

1. If a data item, or its subordinate data items, specifying the BASED ON clause is referenced without explicit pointer, the value indicating the area that can be referenced must be stored in data-name-1.
2. If a data item using the BASED ON clause is specified in the object of a REDEFINES clause, the same BASED ON clause is assumed in the subject of the REDEFINES clause. In the same way, if a data item using the BASED ON clause is specified in the object of a REDEFINES clause, the same BASED ON clause is assumed in the subject of the REDEFINES clause.

5.4.2 BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause replaces zeros in a data item with spaces when the data item contains all zeros.

For details about how to use the examples, refer to "COBOL Syntax Samples", "BLANK WHEN ZERO Clause".

Format

BLANK WHEN { ZERO / ZEROS / ZEROES }

Syntax Rules

1. The BLANK WHEN ZERO clause may be specified in elementary items only.
2. The BLANK WHEN ZERO clause may only be specified in numeric edited data items.
3. The BLANK WHEN ZERO clause may only be specified in unsigned zone decimal numeric data items.
4. ZERO, ZEROS, and ZEROES are synonyms.

General Rules

1. The BLANK WHEN ZERO clause replaces zeros in a data item with spaces when the data item contains all zeros.
2. If the BLANK WHEN ZERO clause is specified in a numeric data item, the category of the numeric data item is assumed to be numeric edited.

5.4.3 CHARACTER TYPE Clause

The CHARACTER TYPE clause specifies format of characters when printing.

Format 1

Specifies the character size of National data items or national edited data items.

[CHARACTER TYPE IS] { MODE-1 / MODE-2 / MODE-3 } [BY mnemonic-name-1]

Format 2

Specifies the character size, pitch, font, rotations, and style of National data items or national edited data items.

CHARACTER TYPE IS mnemonic-name-2

Format 3

Specifies the character size, pitch, font, rotations, and style of National data items or national edited data items of optional data items.

CHARACTER TYPE IS

{ printing-mode-name-1
| [printing-mode-name-2] ... DEPENDING ON data-name-1 }

[OR

{ printing-mode-name-3
| [printing-mode-name-4] ... DEPENDING ON data-name-2 }]

Syntax Rules

- Rules Common to Format 1 and Format 3

  1. The CHARACTER TYPE clause can be specified in data items of any level-number.
  2. The CHARACTER TYPE clause cannot be used in a data item that utilizes a REDEFINES clause at a level-number other than 01 or 77, or a data item which is subordinate to such a data item.
  3. The CHARACTER TYPE clause cannot be specified for a data item that utilizes a TYPEDEF clause nor its subordinates.
  4. The CHARACTER TYPE clause cannot be specified with a data item that utilizes the OCCURS DEPENDING ON clause.

- Rules Common to Format 1 and Format 2

  1. The CHARACTER TYPE clause can only be specified in a group item which utilizes National data items and national edited data items, or a group item which utilizes national edited data items.
     However, when the CHARACTER TYPE clause is specified in a group item that utilizes National data items or national edited data items, the National data items or national edited data items must not depend on redefined items.
  2. If the CHARACTER TYPE clause is specified in a group item, it is applied to all National data items and national edited data items which are subordinate to the group item.
  3. If the CHARACTER TYPE clause is specified both in a group item and a data item that is subordinate to the group item, CHARACTER TYPE clauses having the same meaning must be specified.

- Rules for Format 1

  1. Mnemonic-name-1 must be associated with the following function-name in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

     ```
     HSC, F0202, H0202, F0102, F0201
     ```

  2. If mnemonic-name-1 is associated with the following function-name in the SPECIAL-NAMES paragraph, only MODE-1 or MODE-2 can be specified.

     ```
     HSC, H0202
     ```

- Rules for Format 2

  Mnemonic-name-2 must be associated with function-name starting with the following character-string in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

  ```
  GTA, GTB, GTC, GTD, GTX, GYA, GYB, GYC, GYD, GYX, TA, TB, TC, TD, TX, YA, YB, YC, YD, YX
  ```

- Rules for Format 3

  1. A CHARACTER TYPE clause of format 1 or format 2 must not be specified in a data item that depends on the data item which specified the CHARACTER TYPE clause of format 3 or in a group data item that contains the data item which specified the CHARACTER TYPE clause of format 3.
  2. Printing-mode-name-1 to printing-mode-name-4 must be defined with the PRINTING MODE clause in the SPECIAL-NAMES paragraph.
  3. The CHARACTER TYPE clause with the OR phrase can be specified only in group items.
  4. If the CHARACTER TYPE clause is specified in an elementary item, the following rules must be observed:
     a. Usage of the elementary item must be presentation.
     b. If the elementary item is a National data item or a national edited data item, printing-mode-name-1 and printing-mode-name-2 must not be associated with a PRINTING MODE clause with FOR SOCS in the SPECIAL-NAMES paragraph.
     c. If the elementary item is an alphabetic data item, alphanumeric data item, alphanumeric edited data item, external decimal data item, external Boolean data item, zoned floating-point data item, or numeric edited data item, printing-mode-name-1 and printing-mode-name-2 must not be associated with a PRINTING MODE clause with FOR MOCS in the SPECIAL-NAMES paragraph.
  5. If the CHARACTER TYPE clause is specified in a group item, it is applied to all subordinate presentation data items of the group item. Among the subordinate data items, the CHARACTER TYPE clause is not applied to those which use the CHARACTER TYPE clause of format 3 and subordinate data items of the data item.
  6. If printing-mode-name-2 is specified twice or more in one CHARACTER TYPE clause, all FOR phrases of the PRINTING MODE clause in the SPECIAL-NAMES paragraph must specify the same printing mode names.
  7. If the OR phrase is written, the following rules must be observed:
     a. Printing-mode-name-1 to printing-mode-name-4 must not be associated with a PRINTING MODE that specified FOR ALL in the SPECIAL-NAMES paragraph.
     b. If printing-mode-name-4 is specified twice or more in one CHARACTER TYPE clause, all FOR phrases of the PRINTING MODE clause in the SPECIAL-NAMES paragraph must specify the same printing mode names.
     c. The FOR phrase of the PRINTING MODE clause in the SPECIAL-NAMES paragraph for printing-mode-name-3 or printing-mode-name-4 must not be the same as the FOR phrase of the PRINTING MODE clause in the SPECIAL-NAMES paragraph for printing-mode-name-1 or printing-mode-name-2.
  8. Data-name-5 and data-name-6 can be qualified.
  9. Data-name-5 and data-name-6 must be integer items.
  10. Data-name-5 and data-name-6 must be data items defined in one of the following:
      - WORKING-STORAGE SECTION
      - FILE SECTION
      - CONSTANT SECTION
      - LINKAGE SECTION
      - LOCAL-STORAGE SECTION ([Winx64], [Linux64] and [.NET] only)
  11. Data-name-5 and data-name-6 must not be in a variable position in a record.

12. If the CHARACTER TYPE clause is specified in a record description entry that uses the EXTERNAL clause, data-name-5 and data-name-6 must have the external attribute. The data description entry of data-name-5 and data-name-6 must be written in the same data division.

13. If the CHARACTER TYPE clause is specified in a record description entry that uses the GLOBAL clause, data-name-5 and data-name-6 must be global names. The data description entry of data-name-5 and data-name-6 must be written in the same data division.

General Rules

- Rules for Format 1

1. The CHARACTER TYPE clause of format 1 specifies the character size when National data items or national edited data items are printed.

2. MODE-1 indicates 12-point characters, MODE-2 indicates 9-point characters, and MODE-3 indicates 7-point characters.

   Function-name associated with mnemonic-name-1 has the following meanings.

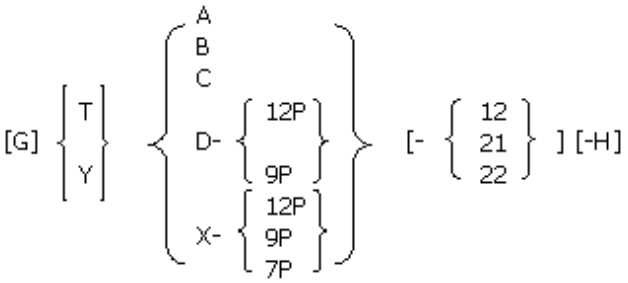
   - HSC: En-size character
   - F0202: Double size character
   - H0202: Double size character of en-size
   - F0102: Tall character
   - F0201: Expanded character

- Rules for Format 2

1. A CHARACTER TYPE clause of format 2 specifies character size, pitch, font, printing direction and style when National data items or national edited data items are printed.

2. Function-name associated with mnemonic-name-2 has the following format and meanings.

   Format



   Meanings

   - Specification of san serif font

     G: San serif font

   - Specification of character printing direction

     T: Vertical printing (Characters are rotated 90 degrees counterclockwise for printing.)

     Y: Horizontal printing (normal printing)

   - Specification of the character size and pitch

     A: 9-point characters are printed by 2 pitch.

     B: 9-point characters are printed by 1.5 pitch.

     C: 9-point characters are printed by 7.5 CPI.

     D-12P: 12-point characters are printed by 6 CPI.

D-9P: 9-point characters are printed by 6 CPI.

X-12P: 12-point characters are printed by character pitch.

X-9P: 9-point characters are printed by character pitch.

X-7P: 7-point characters are printed by character pitch.

- Specification of the character style

  12: Tall character

  21: Expanded character

  22: Double size character

  H: En-size character

a. 12P, 9P, and 7P are synonyms of MODE-1, MODE-2, and MODE-3 of the CHARACTER TYPE clause of format 1, respectively.

b. 12, 21, and 22 are synonyms of function-name F0101, F0201, and F0202, respectively.

c. H is a synonym of function-name HSC. 22-H is a synonym of function-name H0202.

- Rules for Format 3

1. A CHARACTER TYPE clause of format 3 specifies the character size, pitch, font, rotations, and style when optional data items are printed.
2. When the DEPENDING ON phrase is specified, if the value of data-name-5 is n, then the nth printing-mode-name-2 in the list of pointing-mode-names-2 takes effect. In the same way, if the value of data-name-6 is n, then, the nth printing-mode-name-2 in the list of pointing-mode-names-4 takes effect.
3. If the number in the list of printing-mode-names-2 is m, the value of data-name-5 must be 1 to m. In the same way, if the number in the list of printing-mode-names-4 is m, the value of data-name-6 must be 1 to m.
4. When the file connector associated with the file that uses a data item that specifies a CHARACTER TYPE clause is an external file connector, all programs in the run unit that reference the file connector must include the data item which specifies the CHARACTER TYPE clause described in the Rules for Format 3.

- Rules Common to Format 1 to Format 3

1. The CHARACTER TYPE clause is applied when records are printed with the WRITE statement.
2. For details on the character size, pitch, font, rotations, and style, see the topic titled "PRINTING MODE clause" of the "Composition of the Environment Division" section of Chapter 4, the "Environment Division".

F

5.4.4 ENCODING Clause

The ENCODING clause specifies the encoding form of the data item.

The ENCODING clause is specific to [Win32], [Winx64] and [Linux64].

Format

ENCODING IS alphabet-name-1 [OR alphabet-name-2]

Syntax Rules

1. Alphabet-name-1 and alphabet-name-2 must be an alphabet-name associated with any of the following in the ALPHABET clause.
   - SJIS
   - UTF8
   - UTF16
   - UTF16LE

- UTF16BE
- UTF32
- UTF32LE
- UTF32BE

2. Alphabet-name-1 can be specified only in the following items.
   - Alphanumeric data item
   - Alphanumeric edited data item
   - National item
   - National edited item
   - Group item
3. Alphabet-name-2 can only be specified in the group item.
4. When the ENCODING clause is specified in the alphanumeric data item or alphanumeric edited data item, then alphabet-name-1 must be an alphabet-name in the ALPHABET clause, ALPHANUMERIC must be specified explicitly or implicitly.
5. When the ENCODING clause is specified in the National item or National edited item, then alphabet-name-1 is an alphabet-name where NATIONAL is specified in the ALPHABET clause.
6. When the ENCODING clause is specified in the group item, then alphabet-name-1 and alphabet-name-2 must be in accordance with the following rules.
   - Alphabet-name where ALPHANUMERIC is specified explicitly or implicitly with the ALPHABET clause FOR phrase
   - Alphabet-name where NATIONAL is specified with the ALPHABET clause FOR phrase
7. When the ENCODING clause is specified in the group item, then in alphabet-name-2, the ALPHABET clause FOR phrase must not be same as alphabet-name-1.
8. The ENCODING clause cannot be specified in the data items where the TYPEDEF clause is specified and data items are subordinate to those kinds of data items.
9. The ENCODING clause cannot be specified in the data items where the TYPE clause was specified.

General Rules

1. The sizes of the data items specified by the ENCODING clause are as follows.

<table>
<tr><th></th><th>Alphabet name (encoding form)</th><th>Size of data (byte)</th></tr>
<tr><td>Alphanumeric data item
PIC X(n) ENCODING</td><td>SJIS</td><td>n</td></tr>
<tr><td></td><td>UTF8</td><td>n</td></tr>
<tr><td rowspan="3">National data item
PIC N(n) ENCODING</td><td>SJIS</td><td>n*2</td></tr>
<tr><td>UTF16
UTF16LE
UTF16BE</td><td>n*2</td></tr>
<tr><td>UTF32
UTF32LE
UTF32BE</td><td>n*4</td></tr>
</table>

2. When the ENCODING clause is specified in the group item, then the alphabet name, which is specified explicitly or implicitly in the ALPHABET clause, is applied in the alphanumeric data item or alphanumeric edited data item, and the alphabet-name where NATIONAL is specified is applicable to National item or National edited item.

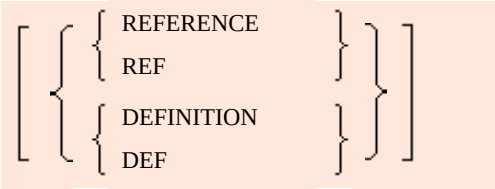
3. When the ENCODING clause is specified, then in elementary items or group items belonging to the group item specified by the ENCODING clause, the specified ENCODING clause is applied in the dependent items.
4. When the alphabet-name associated with SJIS is specified in the ENCODING clause, then the alphabet name associated with UTF8, UTF16, UTF16LE, UTF16BE, UTF32, UTF32LE, or UTF32BE cannot be specified in the same compilation unit.

5.4.5 EXTERNAL Clause

The EXTERNAL clause gives the external attribute to records.

Format

IS EXTERNAL



F The REFERENCE, REF, DEFINITION and DEF phrases are the functions that are specific to [UXP/DS], [Solaris], [LinuxIPF] and [Linux64].

Syntax Rules

1. The EXTERNAL clause can be specified only in the data description entry of a level-number 01 item in the WORKING-STORAGE SECTION.
2. The data-name of a level-number 01 item that specifies the EXTERNAL clause must be unique among other data-names that also specify the EXTERNAL clause within a program.
3. The VALUE clause cannot be used with an external data item, or in any items that are subordinate to the EXTERNAL data item. The VALUE clause can be specified for condition-name entries associated with such data description entries.
4. F The REFERENCE, REF, DEFINITION, and DEF phrases can be specified only for record description entries with the EXTERNAL clause in the WORKING-STORAGE SECTION.
5. The REFERENCE and REF phrases and the DEFINITION and DEF phrases are synonyms, respectively, and can be used interchangeably.
6. In [Linux] and [LinuxIPF], the EXTERNAL clause cannot be specified for BINARY-CHAR/SHORT/LONG/DOUBLE data items.

General Rules

1. The EXTERNAL clause gives an external attribute to a record. External attributes are given to data items that specify the EXTERNAL clause and all subordinate data items.
2. The record that specified the EXTERNAL clause in a record description entry is called an "external data record." Within a run unit, if two or more programs define the same external data record, each record-name of the associated record description entries must be the same, and have the same lengths. External data records of the same record-name reference the same storage.
3. An external data record may be redefined. An external data record may be also be redefined by optional programs in the run unit.
4. F The name of a data description entry specifying the EXTERNAL and the GLOBAL clauses cannot be used as a record-name with the external attribute in a program contained in the program which wrote the data description entry.
5. For all programs in the same run unit that describe the same external record name, EXTERNAL REFERENCE allows you to reference a record defined by one of the following:
    a. An external variable that does not have the extern phrase of the C language.

b. A data description entry that has the EXTERNAL clause with the DEFINITION phrase.

6. For all programs in the same run unit that describe the same external record name, EXTERNAL DEFINITION allows you to define a record referenced or updated by one of the following:
   a. External variables that have the extern phrase of the C language.
   b. A data description entry that has the EXTERNAL clause with the REFERENCE phrase.
7. In a program composing a single run unit, the data-name specified as an object of the entry including the EXTERNAL DEFINITION clause should not be the same as the data-name specified as an object of another entry including the EXTERNAL DEFINITION clause, or the data-name specified as an object of the entry including the EXTERNAL clause without REFERENCE, REF, DEFINITION, or DEF phrase.

5.4.6 GLOBAL Clause

The GLOBAL clause specifies that the data-name is global name.

For details about how to use the examples, refer to "COBOL Syntax Samples", "GLOBAL Clause".

Format

IS GLOBAL

Syntax Rules

1. The GLOBAL clause can be written only in the data description entry of a level-number 01 item in
   - FILE SECTION,
   - WORKING-STORAGE SECTION,
   - LINKAGE SECTION, (F)
   - or CONSTANT SECTION.
2. The data name that specified GLOBAL clause must be unique among other GLOBAL data names within the same data division.
3. The GLOBAL clause must not be specified in a record description entry associated with a file that also uses the SAME RECORD AREA clause.

General Rules

1. The GLOBAL clause specifies that data-name is a global name. The name of the data item that specified the GLOBAL clause and the names of subordinate data items are also global names.
2. A global name can be referenced without redefinition in a program contained directly or indirectly in the program that defined the global name.
3. If the GLOBAL clause is specified in a data item using the REDEFINES clause, the data-name that is the name of the subject of the REDEFINES clause becomes a global name.

5.4.7 JUSTIFIED Clause

The JUSTIFIED clause specifies that as data is transcribed, it is aligned to the rightmost character position of the receiving data item.

For details about how to use the examples, refer to "COBOL Syntax Samples", "JUSTIFIED Clause".

Format

{ JUSTIFIED / JUST } RIGHT

Syntax Rules

1. The JUSTIFIED clause can only be specified in an elementary item.
2. The JUSTIFIED clause can only be specified in the items as follows:
   - alphabetic data item
   - alphanumeric data item
   - national data item

F

3. JUSTIFIED is a synonym of JUST.

General Rules

1. When data is moved to a data item that specifies the JUSTIFIED clause, as data is transcribed it is aligned at the rightmost character position of the receiving item. At this point the receiving item is either truncated or space filled at the left end, depending on whether the sending item is larger or smaller than the receiving item according to the following rules:
   - If the sending data item is larger than the receiving data item, excess characters at the leftmost end of the sending data item are truncated.
   - If the sending data item is shorter than the receiving data item, the leftmost end of the sending data item is padded with spaces.
2. If the JUSTIFIED clause is omitted, standard alignment rules are applied.

5.4.8 OCCURS Clause

The OCCURS clause specifies the number of occurrences of the same data structure.

For details about how to use the examples, refer to "COBOL Syntax Samples", "OCCURS Clause".

Format 1

Specifies a fixed number of occurrences.

```
OCCURS integer-2 TIMES
    [ { ASCENDING  } KEY IS {data-name-2} ... ] ...
      { DESCENDING }
  [INDEXED BY {index-name-1} ... ]
```

Format 2

Specifies a variable number of occurrences.

```
OCCURS [ integer-1 ] TO integer-2 TIMES
       DEPENDING ON data-name-1
    [ { ASCENDING  } KEY IS {data-name-2} ... ] ...
      { DESCENDING }
  [INDEXED BY {index-name-1} ... ]
```

Syntax Rules

- Rules Common to Format 1 and Format 2
  1. The OCCURS clause cannot be specified in a data description entry of level-number 01, 66, 77, or 88.

2. The OCCURS DEPENDING ON clause must not be specified in a data item that is contained within a data item which specifies the OCCURS clause.

F However, this compiler can support an OCCURS DEPENDING ON clause in a data item contained within a data item which specifies the OCCURS clause.

3. Data-name-1 and data-name-2 can be qualified.
4. If the KEY IS phrase is written, data-name-2 must observe the following rules:
   - Data-name-2 must be a data-name of a data description entry which itself specified the OCCURS clause, or a data-name of a data description entry which is subordinate to the data description entry.
   - The data-name specified in data-name-2 may not be repeated in subsequent data-name-2 entries of a single data description entry.
5. Data-name-2 must not be subscripted.
6. F If data-name-2 is not unique in the range of a data description entry which specified the OCCURS clause and a data description entry which depends on the data description entry.
7. For the maximum value of integer-2, see Appendix B, "System Quantitative Restrictions".
8. The OCCURS clause cannot be specified in the data description entry of data-name-2.
9. When the KEY IS phrase is specified, if the data description entry of the data item specified in data-name-2 is subordinate to the data description entry of the item that specifies the OCCURS clause, a data description specifying an OCCURS...KEY IS clause cannot be defined between the data description entry which specified the KEY IS phrase and the data description entry of data-name-2.
10. The data item length of data-name-2 must be 256 bytes or less.
11. Data-name-2 must not be a Boolean data item or index data item.
12. For the maximum number of a sequence of data-name-2, see Appendix B, "System Quantitative Restrictions".
13. The data item that specified index-name-1 as well as all data items subordinate to the data item that specified index-name-1 can be referenced with index-name-1 as a subscript.
14. Storage for index-name is automatically allocated by the compiler and is not dependent on any hierarchy of data.
15. For the maximum number of index-name-1`s, see Appendix B, "System Quantitative Restrictions".
16. F The OCCURS clause with the INDEXED BY phrase must not be specified in a data description entry that also contains the EXTERNAL clause.

- Rules for Format 2

1. Integer-1 must be 0 or more. Integer-2 must be greater than integer-1.

F When integer-1 is omitted, it is assumed that 0 is specified.

2. Data-name-1 must be an integer item.
3. Data-name-1 must not be defined in the range from the first character position of the data description entry which specified the OCCURS clause to the last character position of a record description entry which contains the data description entry.
4. In a record description entry, a data description entry which specified an OCCURS clause of format 2 can be followed by a subordinate data description entries only.

F In this compiler, however, this rule is not enforced.

5. F An OCCURS clause of format 2 must not be specified in a data item in the CONSTANT SECTION.
6. If the OCCURS clause with the KEY IS phrase is specified in a data description that also specifies the EXTERNAL clause, data-name-1 must be a data item having an external attribute. Also, the data description entry specifying the OCCURS clause and the data description entry of data-name-1 must be written in the same data division.

7. If the OCCURS clause with the KEY IS phrase is specified in a data description entry that also specifies the GLOBAL clause, data-name-1 must be a global name. Also, the data description entry containing the OCCURS clause and the data description entry of data-name-1 must be written in the same data division.

General Rules

- Rules Common to Format 1 and Format 2

1. All data description clauses associated with an item whose description includes an OCCURS clause apply to each item described, except for the OCCURS clause itself.
2. If a group item specifying the OCCURS clause contains a binary data item using the SYNCHRONIZED clause, the compiler adds necessary slack bytes to each occurrence of the group item. See the topic titled "Adjustment of Data Boundaries" in section "Concept of Data Description" in Chapter 1, "General Rules" for more information.
3. The number of occurrences of a data item specifying the OCCURS clause is as follows:
   a. In format 1, the value of integer-2 indicates the fixed number of occurrences.
   b. In format 2, the current value of the data item of data-name-1 indicates the number of occurrences.
4. If the KEY IS phrase is specified, the occurrence data must be arranged in ascending order (when ASCENDING is written) or descending order (when DESCENDING is written) according to the value of data-name-2. The order is determined by rules for comparison. These data-names are listed from left to right in the order of the key strength.
5. F If the OCCURS clause with the INDEXED BY phrase is specified in an internal Boolean data item, the SYNCHRONIZED clause must be specified in the internal Boolean data item.

- Rules for Format 2

1. Format 2 indicates that the number of occurrences of the data item which specified the OCCURS clause is variable. The maximum number of occurrences is specified in integer-2. The minimum number of occurrences is specified in integer-1. Format 2 indicates that the size of the data item which specified the OCCURS clause is not variable, but the number of occurrences is variable. A data item that specifies a format-2 OCCURS clause of is called a variable occurrence data item.
2. The value of data-name-1 must be >= integer-1 and <= integer-2 when the data item which specified the OCCURS clause or any subordinate data items are referenced. When the value of data-name-1 changes, the contents of occurrence data larger than the value in data-name-1 are undefined.
3. If a group item contains an table that uses a format 2 OCCURS clause, when the group item is referenced the table data is processed as follows:
   a. When the group item does not contain data-name-1, if the value of data-name-1 is n, the table area from the first to nth elements is processed.
   b. When the group item contains data-name-1 and it is used as a sending item, if the value of data-name-1 is n, the table area from the first to nth elements is processed. When the group item is used as a receiving item, the table area indicated by integer-2 is processed.
4. If an OCCURS clause of format 2 is specified in a record description entry and if the RECORD clause of the associated file description or SORT-MERGE file description entry specifies the VARYING phrase, the record description is a variable length record.
5. If a record description entry contains a data item which utilizes a format 2 OCCURS clause, and the data item is associated with a file description or sort-merge file description entry which specifies the RECORD clause with the VARYING phrase but without the DEPENDING ON phrase, the number of occurrences must be specified in data-name-1 before a RELEASE, REWRITE, or WRITE statement is executed.
6. F A format 2 OCCURS clause must not be specified for an internal Boolean data item which omits the SYNCHRONIZED clause.

5.4.9 PICTURE Clause

The PICTURE clause specifies the category, size, and editing format of elementary items.

For details about how to use the examples, refer to "COBOL Syntax Samples", "PICTURE Clause".

Format

{ PICTURE / PIC } IS character-string

Syntax Rules

1. The PICTURE clause can be specified only for elementary items.
2. A PICTURE character-string is made up of a particular combination of characters included in COBOL character set. The category of elementary items is determined by this combination.
3. Lower case alphabetic characters corresponding to PICTURE symbols A, B, E, N, P, S, V, X, Z, CR, and DB are equivalent to their upper case alphabetic characters in a PICTURE character-string.
4. F In [Win32], [Winx64] and [Linux64], a PICTURE character-string may contain a maximum of 50 characters. In other operation systems, a PICTURE character-string may contain a maximum of 30 characters.
5. The PICTURE clause must be specified for every elementary item except for an index data item, a BINARY-CHAR/SHORT/LONG/DOUBLE data item, internal floating-point data item, pointer data item and the subject of a RENAMES clause.
6. PICTURE and PIC are synonyms.
7. If an asterisk (*), the zero suppression symbol, is specified in a PICTURE character-string, the BLANK WHEN ZERO clause cannot be specified.

General Rules

The category of elementary items is determined by the combination of PICTURE symbols. There are nine categories of elementary items:

- alphabetic character
- numeric character
- alphanumeric character
- alphanumeric edited
- numeric edited
- F national language
- national edited
- Boolean
- external floating-point

1. An elementary item whose category is alphabetic character is called an "alphabetic data item." Rules for specifying an alphabetic data item and its contents are shown below.
   a. A PICTURE character-string that contains only the symbols A.
   b. Usage of an alphabetic data item can be DISPLAY only.
   c. An alphabetic data item must consist of one or more alphabetic characters.
2. An elementary item whose category is numeric character is called a "numeric data item." Rules for defining a numeric data item and its contents are shown below.
   a. The PICTURE character-string is made up of a combination of symbols 9, P, S, and V.

      F In 18-digit compatible operation mode, the number of digits allowed in a PICTURE character-string (the total number with format characters 9 and P) is 1 to 18.

In 31-digit extension operation mode, the number of digits allowed in a PICTURE character-string (the total number with format characters 9 and P) is 1 to 31. 31-digit extension operation mode is available in [Winx64] and [Linux64].

b. The usage of a numeric data item can be

- DISPLAY,
- COMPUTATIONAL,
- BINARY,
- PACKED-DECIMAL,
- or COMPUTATIONAL-5.

c. If signs are not specified in a numeric data item, the contents of a numeric data item must contain one or more numeric characters. If specified, it may also contain operational signs.

d. A numeric data item that does not have digits after the decimal point is called "integer item."

A BINARY-CHAR/SHORT/LONG/DOUBLE data item is also a numeric data item. See the "USAGE Clause" for information about these types.

3. An elementary item whose category is alphanumeric character is called an "alphanumeric data item." Rules for defining an alphanumeric data item and its contents are shown below.

a. The PICTURE character-string is made up of a combination of symbols A, X, and 9. An alphanumeric data item having a PICTURE character-string that contains symbols other than X is handled as one having a PICTURE character-string that contains only X. However, a data item having a PICTURE character-string that contains all A or all 9 is not an alphanumeric data item.

b. The usage of an alphanumeric data item can be DISPLAY only.

c. An alphanumeric data item must consist of one or more characters of the computer character set.

4. An elementary item whose category is alphanumeric edited is called an "alphanumeric edited data item." Rules for defining an alphanumeric edited data item and its contents are shown below.

a. The PICTURE character-string is made up of a combination of symbols A, X, 9, B, 0, and /. This character-string must contain at least one A or X, and at least one B, 0, or /.

b. The usage of an alphanumeric edited data item can be DISPLAY only.

c. An alphanumeric edited data item must consist of at least two characters of the computer character set.

5. An elementary item whose category is numeric edited is called a "numeric edited data item." Rules for defining a numeric edited data item and its contents are shown below.

a. The PICTURE character-string is made up of a combination of symbols 9, P, V, B, /, Z, 0, comma (,), decimal point (.), *, +, -, CR, DB, and currency symbol. This character-string must contain at least one B, /, Z, 0, comma, decimal point, *, +, -, CR, DB, or currency symbol. The symbols P and decimal point cannot both be specified in the PICTURE character-string.

In 18-digit compatible operation mode, the number of digits allowed in a PICTURE character-string (the total number with format characters 9 and P) is 1 to 18.
In 31-digit extension operation mode, the number of digits allowed in a PICTURE character-string (the total number of with format characters 9 and P) is 1 to 31. 31-digit extension operation mode is available in [Winx64] and [Linux64].

b. The usage of a numeric edited data item can be DISPLAY only.

c. The contents of each character position of a numeric edited data item must not conflict with its PICTURE character-string.

6. An elementary item whose category is National is called a "National data item." Rules for defining a National data item and its contents are shown below.

a. The PICTURE character-string is made up of the symbol N only.

b. The usage of a National data item can be DISPLAY only.

c. A National data item must consist of one or more national characters.

7. An elementary item whose category is National edited is called a "national edited data item." Rules for defining a national edited data item and its contents are shown below.

   a. The PICTURE character-string is made up of a combination of N and B. The character-string must contain at least one N and one B.

   b. The usage of a national edited data item can be DISPLAY only.

   c. A national edited data item must consist of two or more national characters.

8. An elementary item whose category is Boolean is called a "Boolean data item." Rules for defining a Boolean data item and its contents are shown below.

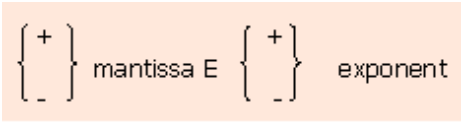
   a. The PICTURE character-string is made up of the symbol 1 only.

   b. The usage of a Boolean data item can be BIT or DISPLAY.

   c. A Boolean data item must consist of a combination of Boolean symbols 0 and 1.

9. An elementary item whose category is external floating point is called an "external floating-point data item." The usage of an external floating-point data item can be DISPLAY only. Rules for defining an external floating-point data item and its contents are shown below.

   a. Format of a PICTURE character-string is shown below.

   b. Signs of the mantissa part must be either one plus sign (+) or one minus sign (-). These signs have the following meanings:

   +: If the value of the data item is positive or zero, append +. If the value of the data item is negative, append "-". The "+" is counted in the size of an elementary item.

   -: If the value of the data item is positive or zero, append a space. If the value of the data item is negative, append "-". The "-" is counted in the size of an elementary item.

   c. Mantissa must be a character-string made up of ., 9, and V. This character-string must contain either V or . and indicates a floating-point data item. These characters have the following meanings:

   9: Indicates one numeric character. The 9 is counted in the size of an elementary item. The number of 9s, that is digit positions of mantissa, must range from 1 to 16.

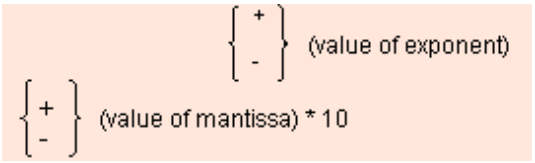
   V: Indicates the position of assumed decimal point. A V is not counted in the size of an elementary item.

   .: Indicates the position of actual decimal point. A . is counted in the size of an elementary item.

   d. An E indicates that an exponent follows. An E is counted in the size of an elementary item.

   e. Exponent must be the character-string 99. 99 is counted in the size of an elementary item.

   f. The value of an external floating-point data item is indicated in the following formula.

   g. The VALUE and BLANK WHEN ZERO clauses must not be specified in an external floating-point data item.

h. The following table lists examples of a PICTURE character-string and the value of an external floating-point data item.

A PICTURE Character-string	External Data Format	Value
+9.9E+99	-5.4E-79	-5.4x(10 to the -79th power)
-99.9(5)E-99	12.34567E 00	12.34567x(10 to the 0th power)
+9(8)VE-99	+12345678E-09	+12345678x(10 to the -9th power)
-V9(5)E+99	-.72000E+76	-0.72x(10 to 76th power)

Rules for a PICTURE Character-string

1. The number of digits of an elementary item counted by the standard data format is called "elementary item size." The elementary item size is determined by the number of characters that indicate character positions.
2. Symbols
   - A,
   - comma (,),
   - X,
   - N,
   - 9,
   - 1,
   - P,
   - Z,
   - *,
   - B,
   - /,
   - 0,
   - +,
   - -,
   - and currency symbol

   can be used repeatedly. If the same symbol is used repeatedly, the number of its occurrences enclosed in parentheses is specified next to the symbol. The number of occurrences must be an unsigned integer other than 0. A sequence of a same symbol is equal to the symbol with the number of its occurrences enclosed in parentheses. For an example, 999 is equal to 9(3).
3. A PICTURE character-string can contain only one of the following:
   - E
   - S
   - V
   - decimal point (.)
   - CR
   - DB
4. For the maximum size of an elementary item, see Appendix B, "System Quantitative Restrictions".

Meaning of PICTURE Symbols

1. An "A" indicates one character position that can contain an alphabetic character or space. An "A" is counted in the size of an elementary item.
2. A "B" indicates
   - one character position that can contain a space
   - or one national character position that can contain a space in a national language.

   A "B" is counted in the size of an elementary item.
3. A "P" indicates an assumed decimal scaling position. A "P" is counted in the size of an elementary item. A "P" may appear in the leftmost or rightmost digit position of a picture clause. If "P" is specified in the leftmost position of a picture clause, the assumed decimal point is assumed to be at the leftmost position of the picture. If "P" is written at the rightmost position of a picture clause, the assumed decimal point is rightmost position of the picture. In both cases, "V," which indicates the position of assumed decimal point, can be omitted.
4. An "S" indicates that an operational sign exists. An "S" must be specified at the beginning of the PICTURE character-string. If the SEPARATE CHARACTER phrase is written in the SIGN clause, "S" is counted in the size of an elementary item. If a phrase other than the SEPARATE CHARACTER phrase is written or the SIGN clause is omitted, "S" is not counted in the size of an elementary item. "S" does not indicate an expression format or the position of the operational sign.
5. A "V" indicates the position of the assumed decimal point. Only one "V" can be specified in a PICTURE character-string. A "V" is not counted in the size of an elementary item, because it does not indicate a character position. If the assumed decimal point is at the rightmost end of a PICTURE character-string ("P" or repetition of "P") and it indicates a digit position or digit alignment position, "V" can be omitted.
6. An "X" indicates one character position. The character position represented by "X" can contain any character from the computer's character set. An "X" is counted in the size of an elementary item.
7. An "N" indicates one national character position. A national character position represented by "N" can contain any national character. An "N" is counted in the size of an elementary item.
8. Each "Z" in a character string indicates that the leftmost numeric character positions, when containing zero are to be replaced with spaces. A "Z" is counted as a character position in the size of an elementary item.
9. A "9" indicates one character position that can contain numeric characters. A "9" is counted in the size of an elementary item.
10. A "1" indicates one Boolean position that can contain one Boolean character. A "1" is counted in the size of an elementary item.
11. A "0" indicates one character position for inserting the numeric character zero. A "0" is counted in the size of an elementary item.
12. A "/" indicates one character position for inserting the character "/." A "/" is counted in the size of an elementary item.
13. A comma (,) indicates one character position for inserting the comma character. A comma is counted in the size of an elementary item.
14. A decimal point (.) indicates the character position where the decimal point should be set. A decimal point also indicates the decimal point position for aligning digits in a data item. A decimal point is counted in the size of an elementary item. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the functions of the decimal point and comma in a picture are exchanged. In this case, the rules related to a decimal point in the PICTURE clause are applied to a comma and the rules related to a comma are applied to a decimal point.
15. Symbols "+", "-", CR, and DB indicate the character position where a symbol indicating positive or negative should be stored. These symbols are called editing sign control symbols. Only one of these symbols can be specified in a PICTURE character-string. The "+" and "-" are each counted as a character position in the size of an elementary item. CR and DB are each counted as two character positions in the size of an elementary item.
16. An "*" (asterisk) indicates that a leading zero string (upper digits of effective numeric characters) is to be replaced with "*". An "*" is counted as a character position in the size of an elementary item.

17. A currency symbol indicates the character position where a currency symbol should be inserted. A "Currency symbol" is a character specified by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If the CURRENCY SIGN clause is omitted, the currency sign becomes the currency symbol. A currency symbol is counted as one character position in the size of an elementary item.

F

18. An "E" indicates the start of the characteristic in an external floating-point data item. An "E" is counted as one character position in the size of an elementary item.

Rules for a Data Item Containing the Symbol "P"

In certain operations that reference a data item whose PICTURE character-string contains the symbol "P", the algebraic value rather than the actual character representation is used as the value of a data item. An algebraic value is a value that assumes the decimal point at the position of "P" and assumes zero at the position of "P." The size of an algebraic value is the number of digit positions represented by the PICTURE character-string, that is the total number of "P" symbols and "9" symbols. These operations are any of the following:

a. An elementary MOVE statement where the sending operand is a numeric data item and its PICTURE character-string contains the symbol "P"

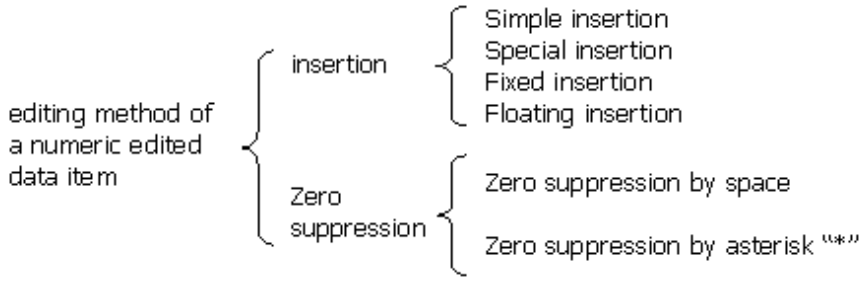
b. When a sending item is a numeric data item whose PICTURE character-string contains the symbol "P"

c. A MOVE statement where the sending operand is a numeric edited data item whose PICTURE character-string contains the symbol "P" and the receiving data item is numeric or numeric edited data

d. In a comparison operation where both operands are numeric data items, and at least one of the operands PICTURE character-string contains the symbol "P"

If a data item's PICTURE character-string contains the symbol "P" and is specified at a place other than mentioned above, the position of "P" is ignored and not counted in the size of the operand.

Editing Rules for an Edited Data Item



There are two editing methods, insertion and zero suppression. The editing method applied to a numeric edited data item is determined by the receiving data item's PICTURE character-string.

Simple insertion of space, "0," or "/" is specified in the PICTURE character-string of an alphanumeric edited data item.

F

Simple insertion of a National space character is specified in a PICTURE character-string of a national edited data item.

Simple insertion

1. In the "simple insertion" method, a comma, space, "0," or "/" is inserted at the position of the insertion character as specified in the PICTURE character-string. The insertion characters in a PICTURE character-strings are the comma, the symbol "B," the symbol "0," and the symbol"/". These insertion characters are called "simple insertion characters." The location of a simple insertion character in a PICTURE character-string indicates the location for the insertion.

2. If a comma (,) is specified at the very end of a PICTURE character-string, the PICTURE clause must be the last clause in the data description entry. Also the PICTURE clause must be directly followed by a period. In the result, ",." appears in the data description entry. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, two consecutive periods ".." appear.

Special insertion

1. In the "special insertion" method, a decimal point is inserted at the position of the insertion character as specified in the PICTURE character-string. If the special insertion method is to be used, the insertion character "." must be specified in the PICTURE character-string. This insertion character is handled as one of the simple insertion characters. A decimal point is not only an insertion character, but also indicates the actual position of the decimal point. Period (.) and "V" are mutually exclusive in one PICTURE clause.

2. If a period (.) is written at the end of a PICTURE character-string, the PICTURE clause must be the last clause in the data description entry. Also the PICTURE clause must be directly followed by a period. In the result, two consecutive periods ".." appear in the data description entry. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, ",." will appear.

Fixed Insertion

1. In the "fixed insertion" method, the value of the data item determines what character is inserted at the position of the insertion character as specified by the data item's PICTURE character-string. The insertion characters used for the fixed insertion method are the currency symbols and the editing sign control symbols (+, -, CR, or DB). These characters are called fixed insertion characters. Only one currency symbol and one editing sign control symbol can be specified in a PICTURE character-string.

2. The position of CR or DB in a PICTURE character-string must be the leftmost end or rightmost end of all character positions counted in the size of an elementary item.

3. The position of a currency symbol must be at the leftmost end of all character positions counted in the size of an elementary item except for cases when the currency symbol is preceded by symbols "+" or "-".

4. The value of a data item determines which characters, listed in the following table, are inserted in the position of editing sign control symbols written in a PICTURE character-string.

Editing Sign Control Symbol in PICTURE Character-string	Character Inserted as a Result of Editing	
	When the Value of a Data Item is Positive or Zero	When the Value of a Data Item is Negative
+	+	-
-	One space	-
CR	Two spaces	CR
DB	Two spaces	DB

Floating Insertion

1. In the "floating insertion" method, two or more consecutive currency symbols, two or more consecutive plus symbols, or two or more consecutive minus symbols are used as insertion characters in a PICTURE character-string. These characters are called "floating insertion characters." The value of the sending item in a MOVE statement cause two or more characters to replace the floating insertion characters. A PICTURE character-string must not contain both a plus and a minus symbol.

2. Simple insertion characters can be written between floating insertion character strings. Simple insertion characters may also be written in the rightmost position of the PICTURE character-string. These simple insertion characters are regarded as a part of the floating insertion character-string.

3. If a currency symbol is written as a floating insertion character, a fixed insertion character CR or DB can be specified at the immediate right of the floating insertion character-string.

4. The leftmost character of a floating insertion character-string indicates the leftmost character position for inserting a floating insertion character. The rightmost character of a floating insertion character-string indicates the rightmost character position for inserting a floating insertion character. The second character from the left side of a floating insertion character-string indicates the rightmost character position for string numeric data in a data item.

5. If the leading digit positions corresponding to the integer part of a PICTURE character-string or all digit positions corresponding to the integer part are insertion characters, the result of floating insertion is as follows:

    a. If the value of the integer part of a data item is zero, one insertion character is inserted in the character position to the right of the decimal point. Spaces are inserted on the left side of the insertion character.

b. If the value of the integer part of a data item is not zero, an insertion character is inserted in the character position to the left of the first non-zero numeric character in the integer part. Spaces are inserted on the left side of the insertion character.

6. If all digit positions of a PICTURE character-string are insertion characters, the result of floating insertion is as follows:

   a. If the value of a data item is zero, the data item is filled with spaces.

   b. If the value of a data item is not zero, one insertion character is inserted in the character position to the left of the decimal point or at the first non-zero character in the integer part. Spaces are inserted on the left side of the insertion character.

7. Currency symbols are inserted in the position of currency symbol insertion characters as specified in a PICTURE character-string. Characters listed in the following table are inserted in the position of the "+" or "-" symbols specified in a PICTURE character-string:

Editing Sign Control Symbol in PICTURE Character-string	Character Inserted as a Result of Editing	
	When the Value of a Data Item is Positive or Zero	When the Value of a Data Item is Negative
+	+	-
-	One space	-

8. To avoid truncation of a receiving numeric edited data item, the number of characters of the receiving item's PICTURE character-string must be greater than the sum of the following:

   a. The number of characters of the sending data item

   b. The number of fixed insertion characters specified in the receiving item's PICTURE character-string

   c. One character for a floating insertion character

   If truncation occurs, the value of the data after truncation is used for editing. For details on the truncation method, see the topic titled "Standard alignment rules" in the "Concept of Data Description" section of Chapter 1, "General Rules".

Zero Suppression

1. "Zero suppression" is the replacement of leading zeros (digits above significant digits) with spaces or asterisks (*). If zero suppression is used, one or more consecutive "Z"s, or one or more consecutive asterisks must be specified in the receiving item's PICTURE character-string. These characters are called "zero suppression characters". Zero suppression characters indicate the digit positions for replacement of a leading zero string. A "Z" specified in a digit position of a PICTURE character-string is replaced with a space, and "*" in a digit position is replaced with "*". "Z" and "*" are mutually exclusive in a PICTURE character-string.

2. Simple insertion characters can be specified between zero suppression character-strings, and simple insertion characters may be specified in the rightmost positions of a PICTURE character-string. These simple insertion characters are regarded as a part of the zero suppression character-string.

3. If the leading digit positions corresponding to the integer part of a PICTURE character-string or all digit positions corresponding to the integer part of a PICTURE character-string are zero suppression characters, the result of zero suppression is as follows:

   a. If the value of the integer part of a data item is zero, the leading zero string up to the left side of the decimal point is replaced with spaces or asterisks (*).

   b. If the value of the integer part of a data item is not zero, the leading zero string up to the left side of the first non-zero numeric character of the integer part is replaced with spaces or asterisks (*).

4. If all digit positions of a PICTURE character-string are zero suppression characters, the result of zero suppression is as follows:

   a. If the value of a data item is zero, the result is as follows:

      - If Z is written as the zero suppression character, the data item is entirely replaced with spaces.

- If * is written as the zero suppression character and simple insertion character decimal point (.) is not specified, the data item is entirely replaced with asterisks (*).
- If * is written as the zero suppression character and the simple insertion character decimal point (.) is specified, digit positions other than the decimal point are replaced with asterisks (*) and the digit position of the decimal point is replaced with a decimal point.

b. If the value of a data item is not zero, the decimal point or the leading zero string up to the left side of the first non-zero numeric character of the integer part is replaced with space(s) or asterisk(s) (*).

Combination of Floating Insertion and Zero Suppression

Floating insertion and zero suppression are mutually exclusive for one numeric edited data item, that is, both kinds of zero suppression cannot be specified. Therefore, when +, -, **,** Z, and the currency symbol are used as floating insertion characters or zero suppression characters, only one type of floating insertion character (+ or -) and one type of zero suppression character (***** or Z) may be specified.

Rules for the Sequence of PICTURE Character-strings

Rules for the sequence of PICTURE character-strings are shown below. The characters that can be specified in a PICTURE character-string depend on the category of the data item. Apply rules for the table based on the category of the data item.

The following explains how to read the table:

- o : The characters shown at the top of the row can be written before (no need to be immediately before) the character shown at the left side of the line.
- - : The characters shown at the top of the row must not be written before the character shown at the left side of the line.
- Currency : Currency symbol
- L : A first or second symbol is written to the left of the decimal point.
- R : A first or second symbol is written to the right of the decimal point.

			First symbol																							
			Simple insertion symbols					Fixed insertion symbols				Zero suppression symbols		Floating insertion symbols				Other symbols							
			B	O	/	,	.	+-		CRDB	Currency	Z*		+-		Currency		9	AX	S	V	P		N	1
								L	R			L	R	L	R	L	R					L	R		
Second Symbol	**Simple insertion symbols**	**B**	o	o	o	o	o	o	-	-	o	o	o	o	o	o	o	o	o	-	o	-	o	o	-
		O	o	o	o	o	o	o	-	-	o	o	o	o	o	o	o	o	o	-	o	-	o	-	-
		/	o	o	o	o	o	o	-	-	o	o	o	o	o	o	o	o	o	-	o	-	o	-	-
		,	o	o	o	o	o	o	-	-	o	o	o	o	o	o	o	o	-	-	o	-	o	-	-
		.	o	o	o	o	-	o	-	-	o	o	-	o	-	o	-	o	-	-	-	-	-	-	-

				First symbol																										
				Simple insertion symbols					Fixed insertion symbols					Zero suppression symbols		Floating insertion symbols				Other symbols										
				B	O	/	,	.	+-		CR DB	Currency		Z*		+-		Currency		9	AX	S	V	P		N	1			
									L	R				L	R	L	R	L	R					L	R					
	Fixed insertion symbols	**+-**	**L**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			R	o	o	o	o	o	-	-	-	o	o	o	-	-	o	o	-	-	-	-	o	o	o	-	-	-	-	-
		CR DB		o	o	o	o	o	-	-	-	o	o	o	-	-	o	o	-	-	-	-	o	o	o	-	-	-	-	-
		Currency		-	-	-	-	-	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Zero suppression symbols	**Z***	**L**	o	o	o	o	-	o	-	-	o	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			R	o	o	o	o	o	o	-	-	o	o	o	-	-	-	-	-	-	-	-	o	-	o	-	-	-	-	-
	Floating insertion symbols	**+-**	**L**	o	o	o	o	-	-	-	-	o	-	-	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
			R	o	o	o	o	o	-	-	-	o	-	-	o	o	-	-	-	-	-	-	o	-	o	-	-	-	-	-
		Currency	**L**	o	o	o	o	-	o	-	-	-	-	-	-	-	o	-	-	-	-	-	-	-	-	-	-	-	-	-
			R	o	o	o	o	o	o	-	-	-	-	-	-	-	-	-	o	o	-	-	-	o	-	o	-	-	-	-
	Other symbols	**9**		o	o	o	o	o	o	-	-	o	o	-	o	-	o	-	o	-	o	o	o	o	-	o	-	-	-	-
		AX		o	o	o	-	-	-	-	-	-	-	-	-	-	-	-	o	o	-	-	-	-	-	-	-	-	-	-
		S		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		V		o	o	o	o	-	o	-	-	o	o	-	o	-	o	-	o	-	o	-	o	-	o	-	-	-	-	-
		P	**L**	o	o	o	o	-	o	-	-	o	o	-	o	-	o	-	o	-	o	-	o	-	o	-	-	-	-	-
			R	-	-	-	-	-	o	-	-	o	-	-	-	-	-	-	-	-	-	-	o	o	-	o	-	-	-	-

			First symbol																							
			Simple insertion symbols					Fixed insertion symbols				Zero suppression symbols		Floating insertion symbols				Other symbols							
			B	O	/	,	.	+-		CR DB	Currency	Z*		+-		Currency		9	AX	S	V	P		N	1
								L	R			L	R	L	R	L	R					L	R		
		N	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	o	-
		1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	o

F

5.4.10 PRINTING POSITION Clause

The PRINTING POSITION clause specifies the column to use when printing.

Format

PRINTING POSITION IS integer-1

[BY positioning-unit-name-1]

Syntax Rules

1. If the PRINTING POSITION clause is specified in a National data item or national edited data item, the CHARACTER TYPE clause must be specified in the data item or in the group item that contains the data item.
2. The PRINTING POSITION clause must not be specified in a data item, or a subordinate data item, which also specifies the OCCURS clause.
3. The OCCURS clause with the DEPENDING ON phrase must not be specified in a record description entry containing a data item which also utilizes the PRINTING POSITION clause.
4. The PRINTING POSITION clause must not be specified in a data item, or in a subordinate data item, with a level number other than 01 or 77 that also specifies the REDEFINED clause.
5. The PRINTING POSITION clause must not be specified in a data item, or its subordinates, that also specifies the TYPEDEF clause.
6. A record description entry containing a data item that specifies the PRINTING POSITION clause must not be the subject or object of the RENAMES clause.
7. Positioning-unit-name-1 must be associated with a literal by using the POSITIONING UNIT clause in the SPECIAL-NAMES paragraph.

General Rules

1. Integer-1 specifies the absolute printing position in a line of printable data items.
2. Positioning-unit-name-1 specifies column units. If positioning-unit-name-1 is omitted, the column units will be 10CPI.
3. Integer-1 specifies the starting print position of a data item to be printed with the leftmost column of a line regarded as 1.

4. If the PRINTING POSITION clause is specified in a group item, integer-1 indicates the leftmost printing position of the group item.
5. Integer-1 must indicate a column to the right of the rightmost column of the data item defined immediately before the printable data items of the record description entry.
6. If the PRINTING POSITION clause is omitted, a data item is printed in the following column:
   - The rightmost data item of a record description entry is printed in the first character position of a print line.
   - The second and subsequent data items in a record description entry are printed directly after the previous data item.

5.4.11 REDEFINES Clause

The REDEFINES clause allows the same computer storage area to be described by different data description entries.

For details about how to use the examples, refer to "COBOL Syntax Samples", "REDEFINES Clause".

Format

level-number [data-name-1 / FILLER] [REDEFINES data-name-2]

Note

Level-number, data-name-1, and FILLER are not part of the REDEFINES clause. They are included in the format only for clarity.

Syntax Rules

1. The REDEFINES clause must directly follow level-number, data-name-1, or FILLER.
2. F In [Linux] and [LinuxIPF], the REDEFINES clause cannot be used with a BINARY-CHAR/SHORT/LONG/DOUBLE data item.
3. The level-numbers of data-name-2 and the subject of the REDEFINES clause must be identical. The level-numbers must not be 66 or 88.
4. F In [Linux] and [LinuxIPF], the data description entry for data-name-2 should not include a BINARY-CHAR/SHORT/LONG/DOUBLE data item.
5. If a file description entry (in the FILE SECTION) contains two or more record description entries, these record description entries are redefined implicitly. The REDEFINES clause must not be specified in the data description entry of a level-number 01 item in the FILE SECTION.
6. The data description entry of data-name-2 must not contain an OCCURS clause. It can be subordinate to a data item which contains an OCCURS clause. The data description entry of data-name-2, a data description entry which contains a REDEFINES clause, and a data description entry which is subordinate to either of these data items must not contain an OCCURS clause with a DEPENDING ON phrase.
7. If a data description entry with a level-number other than 01 contains the REDEFINES clause or data-name-2 is a record-name of an external data record, the number of character positions of data-name-2 must be greater than or equal to the number of character positions of the data item that is the subject of the REDEFINES clause. If a data description entry of level-number 01 contains the REDEFINES clause and data-name-2 is not an external record-name, there is no such a restriction.
8. A data description entry specifying a level-number lower than that of data-name-2 or a data description entry that is the subject of the REDEFINES clause cannot not be written between the data description entry of data-name-2 and the data description entry which contains the REDEFINES clause.

9. A data description entry that contains a REDEFINES clause must follow the data description entry of data-name-2 with no intervening entries that define new storage areas.
10. The same character position can be redefined repeatedly. In this case, data-name-2 must be the data description entry that defined the character position first.

F In this implementation of COBOL, this restriction is lifted.

11. A data description entry that contains a REDEFINES clause must not contain a VALUE clause. However, a condition-name data description entry associated with the data description entry may contain a VALUE clause.
12. Data-name-2 can be subordinate to a data item that specifies the REDEFINES clause.

F 13. If data-name-2 is an internal Boolean data item whose SYNCHRONIZED clause is omitted, the data item to be redefined must also be an internal Boolean data item.

14. The CHARACTER TYPE clause must not be coded in the data description entry of data-name-2. Furthermore, the data description entry of data-name-2 must not be subordinate to a data description entry that has a coded CHARACTER TYPE clause.
15. The data description entry of the data-name-2 must not be a type declaration.
16. The data description entry of the data-name-2 must not specify the TYPE clause.
17. Data-name-1 and data-name-2 must not be a strongly-typed item or subordinate to a strongly-typed item.

General Rules

1. The storage area of data-name-2 is the same size as the data item that specified the REDEFINES clause.
2. If the data-names of two or more data description entries define the same character position, the character position maybe referenced by the data-names of either data description entries.
3. If the subject of the REDEFINES clause or the first elementary item of the subject specifies a SYNCHRONIZED clause, data-name-2 must be defined to match the natural boundary. For example, A must be start at the half word boundary in the following case.

```
02 A PICTURE X(2).
02 B REDEFINES A PICTURE S9(4) BINARY SYNCHRONIZED.
```

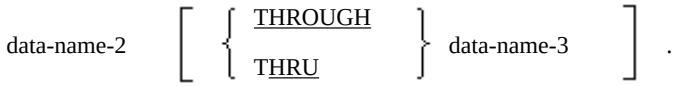
F 4. When the first item in the subject of the REDEFINES clause is a group item, and the group item specifies the TYPE clause, data-name-2 must be aligned on an 8-byte boundary.

5.4.12 RENAMES Clause

The RENAMES clause permits alternative, possibly overlapping groupings of elementary items.

For details about how to use the examples, refer to "COBOL Syntax Samples", "RENAMES Clause".

Format



Note

Level-number and data-name-1 shown in the above format to improve clarity. They are not part of the RENAMES clause.

Syntax Rules

1. A data description entry that contains the RENAMES clause is called a "RENAMES entry." Any number of record description entries can be specified for one record description entry.

2. Data-name-2 and data-name-3 must be the names of elementary items or group items of the logical record and cannot have the same name. Their level-numbers cannot be 77, 88, 01 or 66.
3. RENAMES entries must follow the last data description entry of a record description entry.
4. Data-name-1 must not be used as a qualifier.
5. Data-name-1 can be qualified by the name of the level-number 01 data description entry that contains the RENAMES clause, a file description entry, or a sort-merge file description entry.
6. Data-name-2 and data-name-3 must not specify an OCCURS clause or be subordinate to a data item which specifies an OCCURS clause.
7. Data-name-2 and data-name-3 can be qualified.
8. F If data-name-2 or data-name-3 is not unique in the record description entry containing the RENAMES clause, data-name-2 or data-name-3 must be qualified.
9. A variable occurrence data item cannot be specified between the range of data-name-2 and data-name-3.
10. THRU is a synonym of THROUGH.
11. Data-name-3 must not be subordinate to data-name-2.
12. The leftmost end of data-name-3 must not be any further to the left than the leftmost end of the data-name-2. The rightmost end of data-name-3 must be further right than the rightmost end of data-name-2.
13. F When data-name-3 is specified, data-name-2 and data-name-3 must not be internal Boolean data items.
14. Data-name-2 must not be a variable address item.
15. Data-name-2 and data-name-3 cannot specify the TYPEDEF clause, nor may they be subordinate to a TYPEDEF clause.
16. Data-name-2 and data-name-3 cannot specify the TYPE clause.
17. Data items specifying the TYPE clause cannot be defined between the range data-name-2 and data-name-3.
18. When data-name-2 and data-name-3 are not explicitly qualified and when no ambiguity of reference exists, the implied qualification of the 01 level item is used.

General Rules

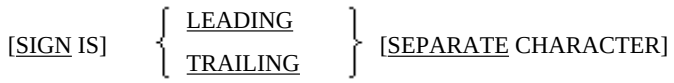
1. If data-name-3 is specified, data-name-1 must be a group item. The group item is composed as follows:
   a. The first subordinate elementary item starts with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item).
   b. The last subordinate ends with data-name-3 (if data-name-3 is an elementary item) or the last elementary item of data-name-3 (if data-name-3 is a group item).
2. If data-name-3 is omitted, the data attributes of data-name-2 become data attributes of data-name-1.
3. F When data-name-3 is specified, any CHARACTER TYPE and PRINTING POSITION clauses that are specified in data-name-2, data-name-3, and in all elementary items between data-name-2 and data-name-3 are ignored in the execution of a WRITE statement specifying the FROM phrase.

5.4.13 SIGN Clause

The SIGN clause specifies the position and mode of representation of the operational sign.

For details about how to use the examples, refer to "COBOL Syntax Samples", "SIGN Clause".

Format



Syntax Rules

1. The SIGN clause can only be specified in a numeric data item whose PICTURE character-string contains symbol "S" or in a group item that contains such a numeric data item.
2. The usage of the data item to which the SIGN clause is applied must be display.

General Rules

1. The SIGN clause specifies the sign position and representation mode of a numeric data item. If the SIGN clause is specified in an elementary item, it is applied to the elementary item. If the SIGN clause is specified in a group item, it is applied to all signed numeric data items that are subordinate to the group item. The sign clause applies only to numeric data whose PICTURE character-string contains the character "S". The "S" indicates the presence, but not the representation, nor necessarily the position of an operational sign.
2. If the SIGN clause is specified in a group item subordinate to a group item that also specifies a SIGN clause, the SIGN clause of the subordinate group item takes precedence.
3. If the SIGN clause is specified in a numeric item subordinate to a group item that also specifies a SIGN clause, the SIGN clause specified in the numeric item takes precedence.
4. If the SIGN clause is not specified for a data item whose PICTURE character-string contains an "S," SIGN IS TRAILING is assumed.
5. When the SEPARATE CHARACTER phrase is omitted:

   a. An operational sign is appended to the leftmost digit position if the LEADING phrase is specified or to the rightmost digit position if TRAILING phrase is specified.

   b. "S" in a PICTURE character-string is not counted in the size of a data item.

   See the User's Guide for information about special subroutines (#DEC88TOFJ and #DECFJTO88) for converting the format of the sign-carrying byte.
6. When the SEPARATE CHARACTER phrase is specified:

   a. An operational sign is appended to the leftmost digit position if the LEADING is specified or to the rightmost digit position if the TRAILING phrase is specified. The character position of the sign does not occupy a digit position.

   b. The symbol S in a PICTURE character-string is counted in the size of a data item.

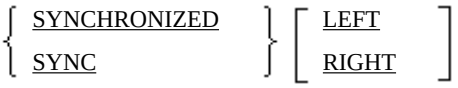
   c. Positive and negative operational signs are "+" and "-" in the standard data format.

   d. If "+" or "-" is not specified as an operational sign, the results are undefined.
7. If a numeric data item that specified a SIGN clause is used for computation or comparison, conversion may take place. The conversion takes place automatically. For the mode of representation of the data item specifying the SIGN clause, see the topic titled "USAGE clause" below.

5.4.14 SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the mapping of an elementary item on a natural boundary in computer storage.

Format



Syntax Rules

1. The SYNCHRONIZED clause can be specified only for an elementary item. In [Win32], [Winx64], [Solaris], [LinuxIPF], [Linux64] and [.NET], the SYNCHRONIZED clause can be also be specified for a group item.

2. SYNC is a synonym of SYNCHRONIZED.

General Rules

1. In [Win32], [Winx64], [Solaris], [LinuxIPF], [Linux64] and [.NET], when the SYNCHRONIZED clause is specified for a group item, the SYNCHRONIZED clause is effective for all elementary items in the group item.
2. The SYNCHRONIZED clause is effective only for elementary items with BINARY, COMPUTATIONAL, COMPUTATIONAL-1(*), COMPUTATIONAL-2(*), COMPUTATIONAL-5(*), INDEX(*), or BIT(*) specified in the USAGE clause. The SYNCHRONIZED clause is ignored if it is specified in other elementary items.
3. An elementary item that specified the SYNCHRONIZED clause is aligned on a natural boundary as shown in the following table.

USAGE Clause Specification	Number of Digits	Natural Boundary
BINARY , COMPUTATIONAL	1 to 4 digits	2-byte (en-size) boundary
	5 to18 digits	4-byte (em-size) boundary
	19 to 31 digits (**)	4-byte (em-size) boundary
COMPUTATIONAL-5	1 to 4 digits	2-byte (en-size) boundary
	5 to18 digits	4-byte (em-size) boundary
	19 to 31 digits (**)	4-byte (em-size) boundary
COMPUTATIONAL-1	-	4-byte (em-size) boundary
COMPUTATIONAL-2	-	8-byte (double size) boundary
INDEX	-	4-byte (em-size) boundary
BIT	-	1-byte boundary

** : For 31-digit extension operation mode - 31-digit extension operation mode is available in [Winx64] and [Linux64].

4. A data item that specified the SYNCHRONIZED clause is aligned on a natural boundary by inserting necessary slack byte(s) or slack bit(s)(*). For details on slack bytes and slack bits(*), see the topic titled "Adjustment of Data Boundaries" in the "Concept of Data Description" section of Chapter 1, "General Rules".
5. Programs execute more efficiently with the SYNCHRONIZED clause specified than without.

F

6. The LEFT and RIGHT phrases are regarded as comments.
7. The SYNCHRONIZED clause may be specified in a data item that utilizes a REDEFINES clause, or in the first subordinate elementary item of a group item that specifies a REDEFINES clause. In this case, the user is responsible for adjusting the data item specified in the object of the REDEFINES clause to the natural boundary specified by the SYNCHRONIZED clause.
8. The compiler aligns level-number 01 data items in the FILE SECTION, WORKING-STORAGE SECTION, LINKAGE SECTION, CONSTANT SECTION, and LOCAL-STORAGE SECTION, and level-number 77 data items in the LINKAGE SECTION on 8-byte (double size) boundaries. The user must be aware of the following:

   a. For records in the FILE SECTION, slack bytes between records must be inserted to start level-number 01 data items at an 8-byte boundary.

   b. Records defined in the WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION and CONSTANT SECTION are aligned automatically.

   c. For records in the LINKAGE SECTION and for level-number 77 data items, data items corresponding to those in the calling program must start on an 8-byte boundary.

   LOCAL-STORAGE section is available in [Winx64], [Linux64] and [.NET].

9. In [Solaris] and [.NET], if the SYNCHRONIZED clause is specified for a group item, it is applied to all subordinate elementary items.

F

* : Extended functions or functions specific to NetCOBOL.

- COMPUTATIONAL-1
- COMPUTATIONAL-2
- COMPUTATIONAL-5
- INDEX
- BIT
- slack bits

F

5.4.15 TYPE Clause

The TYPE clause indicates that a data description entry is a type declaration.

For details about how to use the examples, refer to "COBOL Syntax Samples", "TYPE Clause".

The TYPE clause is specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

Format

TYPE type-name-1

Syntax Rules

1. The description of type-name-1, including its subordinate data, must not contain a TYPE clause that references the subject of the entry.
2. A group item, to which the subject of the entry is subordinate, must not contain the CHARACTER TYPE, PRINTING POSITION, USAGE or SIGN clauses.
3. A subordinate data description entry cannot be written following a data description entry that specifies a TYPE clause.
4. A data item that specifies the TYPE clause cannot be a variable address data item.
5. When type-name-1 is declared with the STRONG phrase, the data item specifying the TYPE clause must be a data item with level-number 01 or a data item that is subordinate to a data item with a type declaration using the STRONG phrase.
6. When type-name-1 is declared with the STRONG phrase, it cannot also specify the TYPEDEF clause.

General Rules

1. The effect of the TYPE clause is as if the data description of type-name-1 were copied replacing the TYPE clause. All aspects of the data description of type-name-1 except for level number, name, GLOBAL and EXTERNAL attributes are applied to the data item that specified the TYPE clause.
2. When type-name-1 is a group item, the level-numbers of subordinate data items are adjusted according to the following rules.
    a. The items that have specified the TYPE clause become a group item whose subordinate items are those of type-name-1`s TYPEDEF definition. The subordinate data items retain the same name, description, and same hierarchy as defined in type-name-1`s TYPEDEF definition.
    b. The level numbers of the subordinate data items are adjusted to preserve the hierarchy as specified in type-name-1`s TYPEDEF definition. The level-number of the resulting hierarchy cannot exceed 49.
    c. The items that have specified the TYPE clause are aligned on an 8-byte boundary as though it were level-number 01 item.
3. When the VALUE clause is specified in the data description of the subject of the TYPE clause, this VALUE clause applies and any VALUE clause specified in the description of type-name-1 is ignored.

Example

An example of using the TYPE clause follows:

```
01 THE-LAYOUT  TYPEDEF.                        *>...1
    03 THE-LAYOUT-GROUP.
        05  ITEM-1   PIC X(10).
        05  ITEM-2   PIC 9(5).
01 A-RECORD.
     03 LAYOUT-1  TYPE THE-LAYOUT.              *>...2
```

The above example results in A-RECORD having the following structure:

```
01 A-RECORD.
    05 THE-LAYOUT-GROUP.
         06  ITEM-1   PIC X(10).
         06  ITEM-2   PIC 9(5).
```

1. This statement defines the structure and hierarchy of THE-LAYOUT record type.
2. THE-RECORD record type is incorporated into the A-RECORD hierarchy. Note that the subordinate data items of THE-RECORD type retain their names, PICTURE character-strings, and relative hierarchy.

F

5.4.16 TYPEDEF Clause

The TYPEDEF clause specifies that a type definition is being declared.

For details about how to use the examples, refer to "COBOL Syntax Samples", "TYPEDEF Clause".

Note

The TYPEDEF clause is a function specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET]. However, STRONG cannot be specified in [.NET].

Format

IS TYPEDEF [STRONG]

Syntax Rules

1. The TYPEDEF clause must not be used in the embedded SQL declarative section.
2. STRONG cannot be specified for elementary items.
3. The TYPEDEF clause cannot be specified with a TYPE clause that specifies the STRONG phrase.
4. When STRONG is specified, a subordinate item must not contain an OCCURS clause with the DEPENDING ON phrase.

General Rules

1. When TYPEDEF is specified, the data description entry becomes a type declaration. The data-name specified as the description entry is regarded as a type-name. Subordinate data description entries are part of the type declaration of this type. The data-names of the subordinate data description entries can only be referenced as the subordinate items of the groups that are defined using type-name.

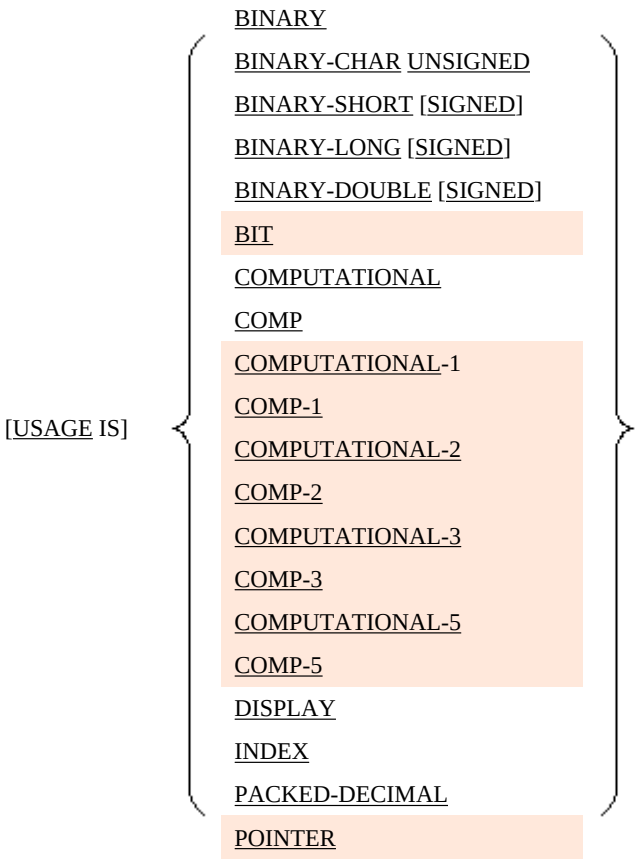
   When one or more of these groups exist, qualification using the name of the group item is necessary. When no groups exist, the data-name of the subordinate entry cannot be referenced, and any resources that are referenced from this name do not exist.
2. The type declaration does not cause allocation of storage.

3. The GLOBAL clause applies to the scope of the type-name. Specification of the data description entry and its subordinate data descriptions are done as if they are written in the data descriptions that are defined using type-name.

5.4.17 USAGE Clause

The USAGE clause specifies the usage of a data item.

Format



BINARY-CHAR, BINARY-SHORT, BINARY-LONG, and BINARY-DOUBLE are specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

Syntax Rules

1. The USAGE clause can be specified in the data description entries of all level-numbers other than 66 or 88.
2. If the USAGE clause is specified in a data description entry of a group item, the USAGE clause may also be specified in data description entries of subordinate elementary items and/or group items. In this case, the USAGE clauses must be identical.
3. Elementary items that specify the following phrases in their USAGE clause, as well as subordinate elementary items in group items must be numeric data items, and their PICTURE character-strings must consist of the symbols P, S, V, and 9 only.
   - BINARY
   - COMPUTATIONAL or COMP
   - COMPUTATIONAL-5 or COMP-5
   - PACKED-DECIMAL
   - COMPUTATIONAL-3, or COMP-3

4. The following paired words are synonyms. Either one may be specified.
   - COMPUTATIONAL and COMP
   - COMPUTATIONAL-1 and COMP-1
   - COMPUTATIONAL-2 and COMP-2
   - COMPUTATIONAL-3 and COMP-3
   - COMPUTATIONAL-5 and COMP-5
5. Elementary items as well as subordinate elementary items in a group item that specify the USAGE IS INDEX clause may be used in the following:
   - SEARCH statements
   - SET statements
   - DISPLAY statements
   - A relation condition
   - A PROCEDURE DIVISION or ENTRY statement that specifies the USING phrase
   - The USING phrase of a CALL statement
6. The following clauses must not be written in a data description entry which contains the USAGE IS INDEX clause or subordinate data description entries of a group item:
   - BLANK WHEN ZERO clause
   - CHARACTER TYPE clause
   - JUSTIFIED clause
   - PICTURE clause
   - PRINTER POSITION clause
   - VALUE clause
7. A condition-name data description entry must not be associated with an elementary item that specifies the USAGE IS INDEX clause.
8. The USAGE IS INDEX clause must not be specified in a data item in the CONSTANT SECTION.
9. Elementary items and subordinate elementary items in a group item that specify the USAGE IS BIT clause must be Boolean data items.
10. A data description entry, or a subordinate data description entry in a group item that specify the COMPUTATIONAL-1, COMP-1, COMPUTATIONAL-2, or COMP-2 phrases in a USAGE clause may not also specify the following:
    - BLANK WHEN ZERO clause
    - CHARACTER TYPE clause
    - JUSTIFIED clause
    - PICTURE clause
    - PRINTING POSITION clause
    - VALUE clause
11. A USAGE IS POINTER clause can be specified in the data description entry of the BASED-STORAGE SECTION, WORKING-STORAGE SECTION, CONSTANT SECTION, LINKAGE SECTION, or LOCAL-STORAGE SECTION only. LOCAL-STORAGE section is available in [Winx64], [Linux64] and [.NET].

12. A data description entry, or a subordinate data description entry in a group item that specifies the POINTER phrase in a USAGE clause may not also specify the following clauses:
    - BLANK WHEN ZERO clause
    - CHARACTER TYPE clause
    - JUSTIFIED clause
    - PICTURE clause
    - PRINTING POSITION clause
    - SIGN clause
13. Only pointer data items can be specified in the BASED ON clause.
14. A pointer data item can only be referenced in the following places in the PROCEDURE DIVISION:
    - The USING phrase or RETURNING phrase of the PROCEDURE DIVISION header
    - The USING phrase of the ENTRY statement
    - The USING phrase or RETURNING phrase of the CALL statement
    - The DISPLAY statement
    - A MOVE statement
    - A relation condition using the IF or EVALUATE statements
15. In [Linux] and [LinuxIPF], the USAGE BINARY-CHAR, BINARY-SHORT SIGNED, BINARY-LONG SIGNED or BINARY-DOUBLE SIGNED can only be specified for elementary items with a level number of 01 or 77.
16. In [Linux] and [LinuxIPF], the USAGE BINARY-CHAR, BINARY-SHORT SIGNED, BINARY-LONG SIGNED or BINARY-DOUBLE SIGNED can only be used in the WORKING-STORAGE, CONSTANT and LINKAGE sections. Items declared with these clauses can only be referenced from the PROCEDURE DIVISION.
17. In [Linux] and [LinuxIPF], condition-name data description entries must not be associated with an elementary item that specifies USAGE IS BINARY-CHAR, BINARY-SHORT, BINARY-LONG, or BINARY-DOUBLE.
18. BLANK WHEN ZERO, JUSTIFIED, and PICTURE clause should not be specified with data items whose usage is BINARY-CHAR UNSIGNED, BINARY-SHORT SIGNED, BINARY-LONG SIGNED or BINARY-DOUBLE SIGNED.

F

CHARACTER TYPE, PRINTING POSITION and REDEFINES clause should not be specified with data items whose usage is BINARY-CHAR UNSIGNED, BINARY-SHORT SIGNED, BINARY-LONG SIGNED or BINARY-DOUBLE SIGNED.

In [Win32], [Winx64], [Solaris], [Linux64] and [.NET], the following clauses cannot be specified in data description entries of subordinate elementary items and/or group items of the above clauses: BLANK WHEN ZERO, JUSTIFIED, and PICTURE. In [Linux] and [LinuxIPF], the following clauses cannot be specified: CHARACTER TYPE, PRINTING POSITION, and REDEFINES clauses.

General Rules

1. If the USAGE clause is specified for a group item, it is applied to all subordinate elementary items.
2. The USAGE clause specifies the manner in which a data item is represented in the computer's storage area.
3. The internal representation of a numeric data item is determined by its USAGE clause. The following table lists the internal representation of numeric data items based on the USAGE clause.

USAGE Claus	SIGN Clause	PICTURE Clause	Value	Internal Representation	Remarks
DISPLAY	None	9(4)	6789	36373839	3: Zone bit

USAGE Claus	SIGN Clause	PICTURE Clause	Value	Internal Representation	Remarks
(zoned decimal)		S9(4)	+6789	36373849	4: Positive operation sign 5: Negative operation sign
			-6789	36373859	
	LEADING	S9(4)	+6789	46373839	
			-6789	56373839	
	TRAILING	S9(4)	+6789	36373849	
			-6789	36373859	
	LEADING SEPARATE	S9(4)	+6789	2B36373839	2B: + of standard data format 2D: - of standard data format
			-6789	2D36373839	
	TRAILING SEPARATE	S9(4)	+6789	363738392B	
			-6789	363738392D	
BINARY, COMP (binary)	-	9(4)	1234	04D2	The leftmost bit indicates an operational sign: 0: positive 1: negative A negative value is represented by twos complement
		S9(4)	+1234	04D2	
			-1234	FB2E	
[UXP/DS] [HP] [Solaris] COMP-5 (binary)	-	9(4)	1234	04D2	The leftmost bit indicates an operational sign: 0: positive 1: negative A negative value is represented by twos complement
		S9(4)	+1234	04D2	
			-1234	FB2E	
[Win16] [Win32] [Winx64] [Linux] [LinuxIPF] [Linux64] [.NET] COMP-5 (binary)	-	9(4)	1234	D204	The leftmost bit indicates an operational sign: 0: positive 1: negative A negative value is represented by twos complement
		S9(4)	+1234	D204	
			-1234	2EFB	
PACKED - DECIMAL (internal decimal)	-	9(4)	1234	01234F	F: Operational sign indicating absolute value C: Positive operational sign D: Negative operational sign
		S9(4)	+1234	01234C	
			-1234	01234D	

-: Combination that cannot be specified

F

4. The internal representation of a Boolean data item is determined by its USAGE clause. The following table lists the internal representation of a Boolean data item by its USAGE clause.

USAGE Clause	PICTURE Clause	Value	Internal Representation	Remarks
DISPLAY (External Boolean)	1(8)	11100101	3131313030313031	Indicated by one character of 0 or 1

USAGE Clause	PICTURE Clause	Value	Internal Representation	Remarks
BIT (Internal Boolean)	1(8)	11100101	E5	Indicated by one bit of 0 or 1

5. If an elementary item or a subordinate elementary item of a group item does not specify a USAGE clause, the USAGE IS DISPLAY clause is assumed. A data item to which a USAGE IS DISPLAY clause is applied is called a "data item whose usage is display."

6. The USAGE IS DISPLAY clause indicates that a data item is represented in a standard data format on the storage device and is adjusted to a character boundary. The contents of a data item with the USAGE IS DISPLAY clause specified (either explicitly or implicitly) are handled in a unit of character positions, or byte. The size of storage occupied by a data item in the standard data format is called "the number of character positions." The number of character positions is equal to the number of bytes.

7. The USAGE IS DISPLAY clause can be specified in

   - alphabetic,
   - alphanumeric,
   - alphanumeric edited,
   - numeric edited,
   - numeric,
   - national language,
   - national edited,
   - Boolean,
   - and external floating-point data items.

   If the USAGE IS DISPLAY clause is specified for a group item, it is applied to all subordinate elementary items of the group.

8. A numeric data item that specifies the USAGE IS DISPLAY clause (either explicitly or implicitly) is called a "zoned decimal data item." Each digit position of the zoned decimal data item is represented by one byte.

9. If symbol "S" is specified in a PICTURE character-string of a zoned decimal data item, an operational sign is represented as follows. If a SIGN clause is omitted, a SIGN IS TRAILING clause is assumed.

   a. If a SIGN IS LEADING clause is specified, an operational sign is included in upper four bits of the first digit position.
   b. If a SIGN IS TRAILING clause is specified, an operational sign is included in lower four bits of the last digit position.
   c. If a SIGN IS LEADING SEPARATE CHARACTER clause is specified, a "+"or "-" sign is stored in the standard data format and placed in the character position directly before the first digit position.
   d. If a SIGN IS TRAILING SEPARATE CHARACTER clause is specified, a "+" or "-" sign is stored in the standard data format and placed after the last digit position.

10. A Boolean data item that specifies the USAGE IS DISPLAY clause (either explicitly or implicitly) is called an "external Boolean data item." Each Boolean position of an external Boolean data item contains a 1 or 0 stored in the standard data format.

11. A numeric data item that specifies

    - USAGE IS COMPUTATIONAL,
    - USAGE IS COMPUTATIONAL-5
    - or USAGE IS BINARY

clauses (either explicitly or implicitly) is called a "binary data item." The USAGE IS COMPUTATIONAL clause is a synonym for the USAGE IS BINARY clause.

F In [UXP/DS], [HP], and [Solaris], the USAGE IS COMPUTATIONAL-5 clause is also a synonym of the above clauses, but in [Win16], [Win32], [Winx64], [Linux], [LinuxIPF], [Linux64] and [.NET], the USAGE IS COMPUTATIONAL-5 clause has a different internal representation.

The size of the storage area allocated to a binary data item is determined by the number of digits specified in the PICTURE clause, as follows:

- 1 to 4 digits: 2 bytes
- 5 to 9 digits: 4 bytes
- 10 to 18 digits: 8 bytes

F
- 19 to 28 digits: 12 bytes (*)
- 29 to 31 digits: 16 bytes (*)

* : For 31-digit extension operation mode - 31-digit extension operation mode is available in [Winx64] and [Linux64].

12. An elementary item that specifies the USAGE IS INDEX clause (either explicitly or implicitly) is called an "index data item." An index data item contains a value corresponding to an occurrence number of a table element. The SET statement, among other statements, can be used to manipulate an index data item. The size of an index data item is 4 bytes except in [Winx64], [LinuxIPF] and [Linux64], where the size of an index data item is 8 bytes.
13. If the USAGE IS INDEX clause is specified at a group item level, all subordinate elementary items of the group item become index data items. However, the group item that specified the USAGE IS INDEX clause does not become an index data item.
14. If a group item containing a subordinate elementary index data item is specified in a MOVE statement or input-output statement, the index data item is not converted.
15. A numeric data item that specifies the USAGE IS PACKED-DECIMAL clause (either explicitly or implicitly) is called a "packed decimal data item." In a packed decimal data item, each byte other than the last byte contains a two-digit number and the lower 4 bits of the last byte contain an operational sign.

F
16. A Boolean data item that specifies the USAGE IS BIT clause (either explicitly or implicitly) is called an "internal Boolean data item." One Boolean character of an internal Boolean data item is represented by one bit of 1 or 0.
17. An elementary item that specifies (either explicitly or implicitly) the USAGE IS COMPUTATIONAL-1 or COMP-1 is called a "single-precision internal floating-point data item" and has a length of 4 bytes.
18. An elementary item that specifies (either explicitly or implicitly) the USAGE IS COMPUTATIONAL-2 or COMP-2 is called a "double-precision internal floating-point data item" and has a length of 8 bytes.
19. An elementary item that specifies (either explicitly or implicitly) the USAGE IS COMPUTATIONAL-1, COMP-1, COMPUTATIONAL-2, or COMP-2 is called an "internal floating-point data item".
20. The internal format of a floating-point item is stored as follows:
    a. The sign of the mantissa is indicated by the leftmost bit.
    b. For a single-precision floating-point number, the exponent occupies the eight bits following the leftmost bit. For a double-precision floating-point number, the exponent occupies the 11 bits following the leftmost bit.
    c. For a single-precision floating-point number, the mantissa occupies the 23 bits following the exponent bits. For a double-precision floating-point number, the mantissa occupies the 52 bits following the exponent bits.
21. The USAGE IS COMPUTATIONAL-3 and USAGE IS COMP-3 clauses are synonymous with the USAGE IS PACKED-DECIMAL clause.
22. An elementary item that specifies the USAGE IS POINTER clause (either explicitly or implicitly) is called a "pointer-data data item". A pointer-data data item can contain a value that represents the address of a storage area. A pointer-data data item has a length of 4 bytes except in [Winx64], [LinuxIPF] and [Linux64], where a pointer-data data item has a length of 8 bytes.

23. A data description entry that specifies USAGE IS BINARY-CHAR, BINARY-SHORT, BINARY-LONG, or BINARY-DOUBLE is called an " BINARY-CHAR/SHORT/LONG/DOUBLE data item".

F

In [Win32], [Winx64], [Solaris], [LinuxIPF], [Linux64] and [.NET], elementary items subordinate to the above data description entry are also called "BINARY-CHAR/SHORT/LONG/DOUBLE data items".

A BINARY-CHAR/SHORT/LONG/ DOUBLE data item is a numeric item and also an integer item and a binary item.

All storage area allocated to BINARY-CHAR/SHORT/LONG/ DOUBLE data items can store numerical data.

The following tables show:

- the size of the storage area allocated to BINARY-CHAR/SHORT/LONG/ DOUBLE data items, and the range of values they can take.
- examples of the internal representation.

Table 5.1 Amount of Memory Allocated and Range of Values for BINARY-CHAR/SHORT/LONG/DOUBLE Data Items

USAGE Clause	Memory Allocated in Bytes	Value Range
BINARY-CHAR UNSIGNED	1	0 <= n < 256 [2**8]
BINARY-SHORT SIGNED	2	-32,768 [-2**15] <= n < 32,768 [2**15]
BINARY-LONG SIGNED	4	-2,147,483,648 [-2**31] <= n < 2,147,483,648 [2**31]
BINARY-DOUBLE SIGNED	8	-9,223,372,036,854,775,808 [-2**63] <= n < 9,223,372,036,854,775,808 [2**63]

Table 5.2 Internal Representation of BINARY-CHAR/SHORT/LONG/DOUBLE Data Items(in [Win32],[Winx64],[Linux],[LinuxIPF], [Linux64] and [.NET])

USAGE Clause	Value	Internal Representation (in hex)	Remarks
BINARY-CHAR UNSIGNED	123	7B	The leftmost bit is part of the number
BINARY-SHORT SIGNED	+1234	D2 04	The leftmost bit indicates an operational sign: 0: positive 1: negative A negative value is represented by 2`s-complement (i.e. invert all the bits of the positive number and add 1)
	-1234	2E FB	
BINARY-LONG SIGNED	+1234	D2 04 00 00	
	-1234	2E FB FF FF	
BINARY-DOUBLE SIGNED	+1234	D2 04 00 00 00 00 00 00	
	-1234	2E FB FF FF FF FF FF FF	

Table 5.3 Internal Representation of BINARY-CHAR/SHORT/LONG/DOUBLE Data Items(in [Solaris])

USAGE Clause	Value	Internal Representation (in hex)	Remarks
BINARY-CHAR UNSIGNED	123	7B	The leftmost bit is part of the number
BINARY-SHORT SIGNED	+1234	04 D2	The leftmost bit indicates an operational sign:
	-1234	FB 2E	

USAGE Clause	Value	Internal Representation (in hex)	Remarks
BINARY-LONG SIGNED	+1234	00 00 04 D2	0: positive 1: negative A negative value is represented by 2`s-complement (i.e. invert all the bits of the positive number and add 1)
	-1234	FF FF FB 2E	
BINARY-DOUBLE SIGNED	+1234	00 00 00 00 00 00 04 D2	
	-1234	FF FF FF FF FF FF FB 2E	

5.4.18 VALUE Clause

The VALUE clause specifies the initial value of a data item or specifies the value of a condition-name.

For details about how to use the examples, refer to "COBOL Syntax Samples", "VALUE Clause".

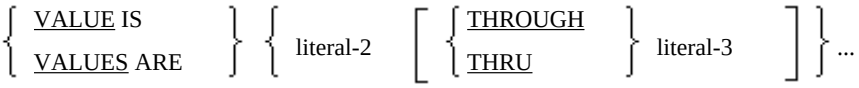
Format 1

Gives the initial values to a data item.

VALUE IS literal-1

Format 2

Specifies the value of condition-name.



Syntax Rules

1. THRU is a synonym of THROUGH.
2. If the VALUE clause is specified in a numeric data item or in condition-name of conditional variable of a numeric data item, literal-1 to literal-3 must be numeric literals. In this case, the value of the numeric literals must not exceed the capacity of the PICTURE clause of the numeric data item.
3. If the VALUE clause is specified in a signed numeric data item or in condition-name of conditional variable which is a signed numeric data item, literal-1 to literal-3 must be signed numeric literals.
4. If the following is used in literal-1 to literal-3, the value must not exceed the capacity of the PICTURE clause.
   - Nonnumeric literal
   - National nonnumeric literal
   - Boolean literal
5. The following table lists examples of values which may be specified in the VALUE clause of a data item whose PICTURE clause contains the symbol "P":

PICTURE Character-string	Values Specified in the VALUE Clause
999PP	0, 100, 200, ..., 99900
PP999	0, .00001, .00002, ..., .00999

6. The VALUE clause must not be specified in the following data description entries. However, the VALUE clause can be specified in a condition-name data description entry associated with the following data description entries.
   a. A data description entry which contains the EXTERNAL clause or a data description entry which is subordinate to a data description entry that specifies the EXTERNAL clause

b. A data description entry which contains a REDEFINES clause or a data description entry which is subordinate to a data description entry that contains a REDEFINES clause

F 7. A THROUGH phrase must not be specified in a Boolean condition-name.

General Rules

- Rules Common to Format 1 and Format 2

1. The VALUE clause must not conflict with other clauses specified in the data description entry of the item or in a data description entry within the hierarchy of the item.
2. If the VALUE clause is specified in a numeric data item, the literal of the VALUE clause must be a numeric literal. For a numeric data item in the following, the literal of the VALUE clause is stored according to the standard alignment rules.
   - WORKING-STORAGE SECTION
   - CONSTANT SECTION
F - LOCAL-STORAGE section ([Winx64], [Linux64] and [.NET] only)
3. If the VALUE clause is specified in an alphabetic, alphanumeric, alphanumeric edited, numeric edited, or group item, the literal of the VALUE clause must be a nonnumeric literal.
F 4. If the VALUE clause is specified in a National data item or national edited data item, the literal of the VALUE clause must be a national nonnumeric literal.
5. If the VALUE clause is specified in a Boolean data item, the literal of the VALUE clause must be a Boolean literal.
6. If the VALUE clause is specified in an internal floating-point data item, the literal of the VALUE clause must be a floating-point literal, numeric literal, or the figurative constant ZERO.
7. If the VALUE clause is specified in a pointer data item, the literal of the VALUE clause must be the figurative constant ZERO.
8. If the VALUE clause is specified in an alphanumeric edited data item or numeric edited data item, the literal of the VALUE clause must be specified in the edited form.
9. If a BLANK WHEN ZERO or JUSTIFIED clause is specified in a data definition that also specifies a VALUE clause, these clauses are ignored.

- Rules for Format 1

1. Not all data descriptions may contain a VALUE clause. The section in which a data item is defined determines whether or not a VALUE clause may be specified:
   a. In the FILE SECTION, the VALUE clause can only be specified in a condition-name data description entry. Initial values of data items in the FILE SECTION are undefined.
   b. In the LINKAGE SECTION, the VALUE clause can only be specified in a condition-name data description entry.
   c. In the WORKING-STORAGE SECTION, the VALUE clause can be specified in a condition-name data description entry or in a data description entry. The VALUE clause in a data description entry of the WORKING-STORAGE SECTION is effective only when the program is in its initial state. A data item that specifies the VALUE clause is initialized by the value of the literal in the VALUE clause. The initial value of a data item that omits the VALUE clause is undefined.
F d. In the LOCAL-STORAGE SECTION, the VALUE clause can be specified in a condition-name data description entry or in a data description entry. When the program is activated, the data item that specifies the VALUE clause is initialized by the value of the literal in the VALUE clause. If the VALUE clause is omitted from the data item, its initial value is undefined. The LOCAL-STORAGE section is available in [Winx64], [Linux64] and [.NET].
   e. In the CONSTANT SECTION, the value of a data item must be specified with the VALUE clause.

2. If the VALUE clause is specified for a group item, the literal of the VALUE clause must be a figurative constant or a nonnumeric literal. The initial value is stored regardless of the format of the subordinate items.
3. The VALUE clause must not be specified on subordinate data items when the group item also specifies a VALUE clause.
4. The VALUE clause must not be specified in a group item whose subordinate data items specify a JUSTIFIED, SYNCHRONIZED, or a USAGE clause other than USAGE IS DISPLAY.
5. If the VALUE clause is specified in a variable occurrence data item, a group item whose subordinate data items contain a variable occurrence data item, or data item which is subordinate to a variable occurrence data item, the size of the variable occurrence data item is regarded as the maximum size of the variable occurrence data item. If the VALUE clause has been specified for a data item that also specified the OCCURS DEPENDING ON clause, the VALUE clause is ignored.
6. If the VALUE clause is specified for a data item which also specifies the OCCURS clause or a subordinate data item to an OCCURS clause, the value specified in the VALUE clause is stored in each repetition of the related data item.

- Rules for Format 2

1. A VALUE clause of format 2 may be specified only in a condition-name data description entry.
2. In a condition-name data description entry, condition-name and a VALUE clause of format 2 must be used. The category of the literal used in the VALUE clause is determined by category of the conditional variable.
3. If the THRU phrase is coded
   a. If the condition variable is a group item or if it is an elementary item in the data category of alphabetic items, alphanumeric data items, alphanumeric edited data items, national items, national edited items, or numeric edited data items, the following condition must be satisfied: in a comparison between the character in each character position of literal-2 and the character in the corresponding position of literal-3 according to the collating sequence, literal-2 is less than that of literal-3.
   b. If the condition variable is an elementary item of a category not among the above categories, literal-2 and literal-3 will be compared using their algebraic values, and literal-2 must be less than literal-3.

F

5.5 Screen Data Description Entry

A screen data description entry defines a screen item. It defines the mapping, display attributes, size, and input-output attributes of a screen item. A screen data description entry is defined in the SCREEN SECTION.

There are two kinds of screen items: elementary screen items and group screen items. "Elementary screen items" are the various independent areas on the screen. A "group screen item" is a set of several elementary items.

There are four types of elementary screen items. The type of an elementary screen item is determined by the input-output attribute. The four types of elementary screen items are: "literal item," "input item," "output item," and "update item."

Note

The screen data description entry cannot be used in [Linux], [LinuxIPF], [Linux64] and [.NET].

Format 1

Defines a screen group item

level-number

[data-name-1
  FILLER]

[AUTO clause]

[BACKGROUND-COLOR clause]
[BLANK SCREEN clause]
[FOREGROUND-COLOR clause]
[FULL clause]
[REQUIRED clause]
[SECURE clause]
[SIGN clause]
[USAGE clause].

Format 2

Defines a literal item.

level-number
[data-name-1 | FILLER]
[BACKGROUND-COLOR clause]
[BELL clause]
[BLANK LINE clause | BLANK SCREEN clause]
[BLINK clause]
[COLUMN NUMBER clause]
[ERASE clause]
[FOREGROUND-COLOR clause]
[HIGHLIGHT clause | LOWLIGHT clause]
[LINE NUMBER clause]
[REVERSE-VIDEO clause]
[UNDERLINE clause]
VALUE clause.

Format 3

Define an input, output, or update item.

level-number
[data-name-1 | FILLER]
[AUTO clause]
[BACKGROUND-COLOR clause]
[BELL clause]
[BLANK LINE clause | BLANK SCREEN clause]
[BLANK WHEN ZERO clause]

[BLINK clause]

[COLUMN NUMBER clause]

[ERASE clause]

[FOREGROUND-COLOR clause]

[FULL clause]

[HIGHLIGHT clause]
[LOWLIGHT clause]

[JUSTIFIED clause]

[LINE NUMBER clause]

[PICTURE clause]

[REQUIRED clause]

[REVERSE-VIDEO clause]

[SECURE clause]

[SIGN clause]

[UNDERLINE clause]

[USAGE clause].

Syntax Rules

1. A screen data description entry can only be specified in the SCREEN SECTION.
2. Level-numbers must be between 01 and 49
3. If data-name-1 or FILLER is specified, it must directly follow the level-number. Other clauses may be specified in any order.
4. If data-name-1 or FILLER is omitted, FILLER is assumed.
5. If the level-number is 01, data-name-1 must be specified.
6. A group screen item can contain group screen items or elementary screen items. Group screen items must be defined using format 1. Elementary screen items may be defined using format 2 or format 3.
7. In format 2 and format 3, at least one of the following clauses must be specified: BELL, BLANK LINE, BLANK SCREEN, COLUMN NUMBER, or LINE.
8. A clause may be specified in a group screen item and can be specified for elementary screen items. If the clauses conflict, the clause specified in a screen item at the lowest level in the hierarchical structure takes precedence.
9. A screen item can only be referenced in an ACCEPT or DISPLAY statement of the screen handling module.

General Rules

1. Distinction between an input, output, or update item of format 3 is determined by the PICTURE clause.
2. If a SECURE clause is specified in a group screen item, it is applied to all subordinate input items of the group screen item.
3. If one of the following clauses is specified in a group screen item, the clause is applied to all subordinate input and update items of the group screen item:
   - AUTO clause
   - BACKGROUND-COLOR clause
   - BLANK SCREEN clause
   - FOREGROUND-COLOR clause

- FULL clause
- REQUIRED clause

4. Do not specify a value in the LINE NUMBER or COLUMN NUMBER clauses that may lead to overlapping of areas on the screen. Further, do not specify line or column numbers that exceed the physical dimensions of the screen.

5.5.1 AUTO Clause

The AUTO clause automates movement of the cursor to input and update items.

Format

AUTO

Syntax Rule

The AUTO clause can be specified in an elementary or group screen item other than a literal item.

General Rules

1. If the AUTO clause is specified in a group screen item, the AUTO clause is applied to all subordinate input and update items of the group screen item.
2. During execution of an ACCEPT statement for a group screen item, when the last character is input for an input or update item, the AUTO clause causes the cursor to automatically move to the next input or update item.
3. The AUTO clause is ignored when a DISPLAY statement is executed.

5.5.2 BACKGROUND-COLOR Clause

The BACKGROUND-COLOR clause specifies the background color of a screen item.

Format

BACKGROUND-COLOR IS integer-1

Syntax Rules

1. The BACKGROUND-COLOR clause can be specified with any screen item.
2. The value of integer-1 may be 0 to 7.

General Rules

1. If the BACKGROUND-COLOR clause is specified with a group screen item, it is applied to all subordinate elementary screen items of the group screen item.
2. The BACKGROUND-COLOR clause is effective only for a color screen.
3. Integer-1 specifies the value that indicates the background color of a screen item. The following table lists background colors corresponding to the value of integer-1.

Integer-1	Background Color
0	Black
1	Blue
2	Green
3	Cyan
4	Red

Integer-1	Background Color
5	Magenta
6	Brown or yellow
7	White

4. If the BACKGROUND-COLOR clause is omitted, the background color is black.
5. The BACKGROUND-COLOR clause is applied when DISPLAY and ACCEPT statements are executed. A screen item with an effective BACKGROUND-COLOR clause is displayed in the color specified in integer-1.
6. If a screen item with the BLANK SCREEN and BACKGROUND-COLOR clauses is displayed with a DISPLAY statement, the default value of the background color changes to the color specified with the BACKGROUND-COLOR clause.

5.5.3 BELL Clause

The BELL clause specifies the sounding of the audible tone each time the item containing the clause is displayed.

Format

BELL

Syntax Rule

The BELL clause can be specified only with an elementary screen item.

General Rule

When a screen with the BELL clause is displayed an audible tone sounds.

5.5.4 BLANK LINE Clause

The BLANK LINE clause causes clearing of a display line before a screen item is displayed.

Format

BLANK LINE

Syntax Rule

The BLANK LINE clause may only be specified in an elementary screen item.

General Rules

1. When a screen item with the BLANK LINE clause is displayed, the display line of the screen item is cleared from the left end to the right end before the screen item is displayed.
2. The BLANK LINE clause is ignored when an ACCEPT statement is executed.

5.5.5 BLANK SCREEN Clause

The BLANK SCREEN clause clears the entire screen before a screen item is displayed.

Format

BLANK SCREEN

Syntax Rule

The BLANK SCREEN clause can be specified with any type of screen item.

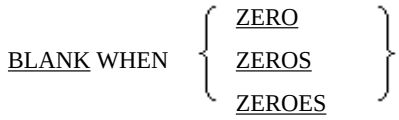
General Rules

1. If the BLANK SCREEN clause is specified with a group screen item, it is applied to all subordinate elementary items of the group screen item.
2. When a screen item with a BLANK SCREEN clause is displayed with a DISPLAY statement, the entire screen is cleared before the screen item is displayed, and the cursor is placed at line 1 column 1.
3. If the BLANK SCREEN and BACKGROUND-COLOR clauses are used together, the default value of the background color will be changed. For the default value of the background color, see the topic titled "BACKGROUND-COLOR clause" above.
4. If the BLANK SCREEN and FOREGROUND-COLOR clauses are used together, the default value of the foreground color will be changed. For the default value of the foreground color, see the topic titled "FOREGROUND-COLOR clause" below.
5. The BLANK SCREEN clause is ignored when an ACCEPT statement is executed.

5.5.6 BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause specifies that spaces should be displayed when the value of a screen item is zero.

Format



Syntax Rules

1. The BLANK WHEN ZERO clause can only be specified in a numeric or numeric edited elementary screen item.
2. If the BLANK WHEN ZERO clause is specified in a numeric screen item, the usage of the numeric screen item must be display.
3. ZERO, ZEROS, and ZEROES are synonyms.

General Rules

1. When the value of a screen item is zero, the BLANK WHEN ZERO clause causes the screen item to display space(s) instead of zero(s).
2. If the BLANK WHEN ZERO clause is specified in a numeric screen item, the category of the item is regarded as numeric edited.
3. If the BLANK WHEN ZERO clause is specified with an input item, it will be ignored.
4. The BLANK WHEN ZERO clause is ignored when an ACCEPT statement is executed.

5.5.7 BLINK Clause

The BLINK clause causes the screen item to blink when it appears on the screen.

Format

BLINK

Syntax Rule

The BLINK clause may only be specified with an elementary item.

General Rules

1. When a screen item with the BLINK clause is displayed, the contents of the screen item will blink.
2. The BLINK clause is ignored when an ACCEPT statement is executed.

5.5.8 COLUMN NUMBER Clause

The COLUMN NUMBER clause specifies the column at which the screen item starts on the screen.

Format

```
COLUMN NUMBER IS [PLUS] { identifier-1 }
                        { integer-1    }
```

Syntax Rules

1. The COLUMN NUMBER clause can only be specified with an elementary screen item.
2. identifier-1 must be an unsigned integer item.

General Rules

1. The COLUMN NUMBER clause specifies the starting column number at which the screen item is displayed or input.
2. If PLUS is omitted, the column number of the first character of the screen item on the screen is the value of identifier-1 or integer-1. The meaning of the value in identifier-1 or integer-1 depends on whether or not the ACCEPT or DISPLAY statement specifies the AT COLUMN NUMBER clause. If the value of identifier-1 or integer-1 is n, the screen item column number is mapped as follows:
    a. If AT COLUMN NUMBER phrase is omitted in a DISPLAY or ACCEPT statement, the screen item is placed at column "n" of the physical screen.
    b. When an AT COLUMN NUMBER phrase is specified with a DISPLAY or ACCEPT statement, the value of the AT COLUMN NUMBER phrase is added to the column number specified in indentifier-1 or integer-1, and the sum specifies the starting column number for the screen item. For example, if the AT COLUMN NUMBER specifies column "x" then the starting column number for the screen item is column x+n.
3. If PLUS is specified, the relative column number from the right end of the previous screen item, regardless of whether the preceding screen item is displayed or accepted, is added to the value in identifier-1 or integer-1 and this value specifies the starting position of the screen item on the screen.
4. A screen item which specifies the COLUMN NUMBER clause with the PLUS phrase cannot be the first screen item of a screen that is the subject of an ACCEPT or DISPLAY statement.
5. The following clauses are assumed in a screen item which omits the COLUMN NUMBER clause:
    a. If a LINE NUMBER clause is specified in the screen item, "COLUMN NUMBER 1" is assumed.
    b. If a LINE NUMBER clause is not specified in the screen item, when there is a preceding screen item, "COLUMN NUMBER PLUS 1" is assumed.

5.5.9 ERASE Clause

The ERASE clause clears part of a line or part of a screen before the screen item is displayed.

Format

ERASE { EOL / EOS }

Syntax Rule

The ERASE clause can only be specified with an elementary screen item.

General Rules

1. When a screen item which specifies the ERASE clause is to be displayed, one of the following screen areas is cleared before the screen item is displayed:
   a. If the ERASE EOL clause is specified, the display line of the screen item is cleared from the first column of the display area of the screen item to the right end of the screen item.
   b. If the ERASE EOS clause is specified, the screen is cleared from the first column of the display line of the screen item to the end of the screen.
2. If one of the following screen items is specified in a DISPLAY statement, only the area on the screen corresponding to the screen item is changed by the DISPLAY statement. Other areas on the screen remain the same as before execution of the DISPLAY statement.
   a. When a group screen item is specified in a DISPLAY statement and an elementary screen item with an effective BLANK clause (BLANK SCREEN clause or BLANK LINE clause) and ERASE clause is not subordinate to the group screen item.
   b. When an elementary screen item with an ineffective BLANK clause (BLANK SCREEN clause or BLANK LINE clause) and ERASE clause is specified in a DISPLAY statement
3. The ERASE clause is ignored when an ACCEPT statement is executed.

5.5.10 FOREGROUND-COLOR Clause

The FOREGROUND-COLOR clause specifies the foreground color of a screen item.

Format

FOREGROUND-COLOR IS integer-1

Syntax Rules

1. The FOREGROUND-COLOR clause can be specified in an optional screen item.
2. The value of integer-1 must be from 0 to 7.

General Rules

1. If the FOREGROUND-COLOR clause is specified in a group screen item, it is applied to all the subordinate elementary screen items.
2. The FOREGROUND-COLOR clause is effective only for a color screen.
3. integer-1 specifies the value which indicates the foreground color of a screen item. The following table lists foreground colors corresponding to the value of integer-1.

Integer-1	Foreground Color
0	Black
1	Blue
2	Green
3	Cyan

Integer-1	Foreground Color
4	Red
5	Magenta
6	Brown or yellow
7	White

4. If the FOREGROUND-COLOR clause is omitted, the foreground color is white.
5. The FOREGROUND-COLOR clause is effective when DISPLAY and ACCEPT statements are executed. A screen item with an effective FOREGROUND-COLOR clause is displayed in the color specified in integer-1.
6. When a screen item that specifies both the BLANK SCREEN and FOREGROUND-COLOR clauses is displayed with a DISPLAY statement, the default value of the foreground color changes to the color specified with the FOREGROUND-COLOR clause.

5.5.11 FULL Clause

The FULL clause specifies that the operator must either leave the screen item completely empty or must fill it completely with data.

Format

FULL

Syntax Rules

1. The FULL clause can be specified in an elementary screen item other than literal item or in a group screen item.
2. If the FULL clause is specified in an elementary screen item, there must be no JUSTIFIED clause.

General Rules

1. If the FULL clause is specified in a group screen item, the FULL clause is applied to all subordinate input items and update items of the group screen item.
2. When inputting data, via an ACCEPT statement, to a screen item specifying the FULL clause, the cursor is placed in the first character position of the screen item, and ACCEPT terminator keystrokes are ignored until the following occurs:
    a. If the screen item is an alphanumeric screen item or alphanumeric edited screen item, until characters other than a space are input in the first and last character positions of the screen item or until the entire screen item is filled with spaces.
    b. If the screen item is a National screen item or National edited screen item, until national nonnumeric characters other than a space are input in the first and last national nonnumeric character positions of the screen item area or until the entire screen item area is filled with National spaces.
    c. If the screen item is a numeric screen item or numeric edited screen item, until numeric characters are input in all digit positions of the screen item. For a numeric edited screen item which specifies zero suppression, however, there is no need to input zeros above the significant digits.
3. If a function key is used to terminate the input operation, rule 2 above is ignored.
4. The FULL clause is ignored when a DISPLAY statement is executed.

5.5.12 HIGHLIGHT Clause

The HIGHLIGHT clause specifies highlighting of a screen item.

Format

HIGHLIGHT

Syntax Rule

The HIGHLIGHT clause can be specified only in an elementary screen item.

General Rules

1. The HIGHLIGHT clause is effective when DISPLAY and ACCEPT statements are executed. A screen item that specifies the HIGHLIGHT clause is highlighted.
2. If the screen item specifies both the HIGHLIGHT and BACKGROUND-COLOR clauses, the background color of the screen item is highlighted. In the same way, if the screen item specifies both the FOREGROUND-COLOR and the HIGHLIGHT clauses, the foreground color of the screen item is highlighted.

5.5.13 JUSTIFIED Clause

The JUSTIFIED clause specifies alignment of the data to the right side of a screen item.

Format

{ JUSTIFIED / JUST } RIGHT

Syntax Rules

1. The JUSTIFIED clause can be specified only in an alphabetic, alphanumeric, or National screen item.
2. JUST is a synonym of JUSTIFIED.

General Rules

1. The JUSTIFIED clause specifies that data in a screen item should be aligned to the right end of the screen item. If the JUSTIFIED clause is specified in a screen item, data in the sending area is stored adjusted to the right end of the screen item. If the sizes of the sending data item and screen item are different, the following processing is done when the data is transferred:
    a. If the sending data item is larger than the screen item, the leftmost characters of the sending data item are truncated to fit the screen item.
    b. If the sending data item is smaller than the screen item, the unused character positions at the left end of the receiving data item are filled with spaces.
2. If the JUSTIFIED clause is omitted, the standard alignment rules for storing data in an elementary item are applied.
3. If the JUSTIFIED clause is specified in an input item, it is ignored.
4. The JUSTIFIED clause is ignored when an ACCEPT statement is executed.

5.5.14 LINE NUMBER Clause

The LINE NUMBER clause specifies the line to which a screen item is mapped.

Format

LINE NUMBER IS [PLUS] { identifier-1 / integer-1 }

Syntax Rules

1. The LINE NUMBER clause can only be specified in an elementary screen item.
2. Identifier-1 must be an unsigned integer item.

General Rules

1. The LINE NUMBER clause specifies the display line number of a screen item when the screen item is displayed or input.
2. If PLUS is omitted, the column number of first character of the screen item on the screen is identifier-1 or integer-1. The meaning of the value in identifier-1 or integer-1 depends on whether or not the ACCEPT or DISPLAY statement specifies the AT LINE NUMBER clause. If the value of identifier-1 or integer-1 is n, the screen item line number is mapped as follows:
    a. If AT LINE NUMBER phrase is omitted in a DISPLAY or ACCEPT statement, the screen item is placed at column "n" of the physical screen.
    b. When an AT LINE NUMBER phrase is specified with a DISPLAY or ACCEPT statement, the value of the AT LINE NUMBER phrase is added to the line number specified in indentifier-1 or integer-1, and the sum specifies the starting line number for the screen item. For example, if the AT LINE NUMBER specifies column "x" then the starting line number for the screen item is column x+n.
3. If PLUS is specified, the relative line number from the right end of the previous screen item, regardless of whether the preceding screen item is displayed or accepted, is added to the value in identifier-1 or integer-1 and this value specifies the starting line number of the screen item on the screen.
4. The LINE NUMBER clause with the PLUS phrase cannot be specified as the first screen item.
5. A screen item that omits the LINE NUMBER clause is mapped in the following places:
    a. If there is no preceding screen item, the screen item is mapped to the first line of the physical screen.
    b. If there is a preceding screen item, the screen item is mapped to the same line number as the preceding screen item.

5.5.15 LOWLIGHT Clause

The LOWLIGHT clause specifies dimming of a screen item.

Format

LOWLIGHT

Syntax Rule

The LOWLIGHT clause can be specified only in an elementary screen item.

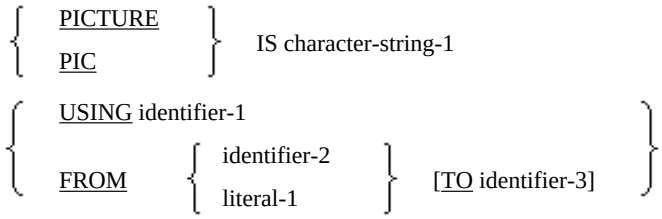
General Rules

1. The LOWLIGHT clause is effective when both DISPLAY and ACCEPT statements are executed. A screen item that specifies the LOWLIGHT clause is dimmed.
2. In a system that has only two display intensities, standard intensity and low intensity are the same.

5.5.16 PICTURE Clause

The PICTURE clause specifies the category of an elementary screen item, the edit format, and associates a data item with an elementary screen item for interchanging data on the screen.

Format



Syntax Rules

1. A PICTURE clause can be specified only in an elementary screen item other than a literal item. The PICTURE clause and VALUE clause are mutually exclusive.
2. The identifier-1, identifier-2 and identifier-3 must be data items defined in the following:
   - FILE SECTION
   - WORKING-STORAGE SECTION
   - LINKAGE SECTION
   - BASED-STORAGE SECTION

   A data item defined in the CONSTANT SECTION can also be specified in identifier-2.
3. One of following must also be specified in a PICTURE clause:
   - USING phrase only
   - FROM phrase only
   - TO phrase only
   - FROM phrase and TO phrase
4. The PICTURE clause, the USING phrase, the FROM phrase and the TO phrase can be specified in any order. Other clauses can appear between a PICTURE clause with the USING phrase, between a PICTURE clause with the FROM phrase, or between a PICTURE clause with the TO phrase.
5. PIC is a synonym of PICTURE.

General Rules

1. The category of an elementary screen item is specified by the contents of character-string-1. There are seven categories of elementary screen items: alphabetic, numeric, alphanumeric, alphanumeric edited, numeric edited, national language, and National edited. Each category of elementary screen items is called "alphabetic screen item", "numeric screen item", "alphanumeric screen item", "alphanumeric edited screen item", "numeric edited screen item", "national screen item", and "national edited screen item". The rules for character-string-1 are the same as those for a PICTURE character-string of a data description entry. For details, see the topic titled "PICTURE clause" in the "Data Description Entry" section of this chapter. However, if a numeric screen item is defined, the usage of the item is display and S must be specified in character-string-1.
2. If data is to be transferred from a data item to a screen item and displayed on the screen, the FROM phrase must be specified. A screen item which specifies only a FROM phrase is called an "output item".
3. If input data is to be transferred from a screen item to a data item, the TO phrase must be specified. An item which specifies only a TO phrase is called an "input item".
4. If data is to be transferred from a data item to a screen item, edited on the screen, and then stored in another data item, both the FROM and TO phrases must be specified. If data is to be transferred to a data item, updated on the screen, and stored in the same data item, the USING phrase must be specified. In both cases, such a screen item is called an "update item."
5. The number of character positions of identifier-1, identifier-2, identifier-3, and literal-1 need not be the same as the number of character positions of the related screen item.

6. Transfer of data between a screen item and identifier-1, identifier-2, identifier-3, or literal-1 is done according to the normal rules for a MOVE statement.
7. The FROM phrase is only applicable when a DISPLAY statement is executed.
8. The TO phrase is only applicable when an ACCEPT statement is executed.
9. The USING phrase is applicable when both ACCEPT and DISPLAY statements are executed.

5.5.17 REQUIRED Clause

The REQUIRED clause specifies that input in a screen item is required.

Format

REQUIRED

Syntax Rule

The REQUIRED clause can be specified in elementary screen items or group screen items other than literal items.

General Rules

1. If the REQUIRED clause is specified in a group screen item, it is applied to all of the group's subordinate input and update items.
2. When inputting data, via an ACCEPT statement, to a screen item specifying the REQUIRED clause, the cursor is placed in the first character position of the screen item and ACCEPT terminator keystrokes are ignored until the following occur:
    a. If the screen item is an alphanumeric screen item or alphanumeric edited screen item, until one or more characters other than a space are input.
    b. If the screen item is a National screen item or National edited screen item, until one or more national nonnumeric characters other than a space are input.
    c. If the screen item is a numeric screen item or numeric edited screen item, until a value other than zero is input.
3. If a function key is used to complete an input operation, the rule 2 above is ignored.
4. The REQUIRED clause is only applicable when a DISPLAY statement is executed.

5.5.18 REVERSE-VIDEO Clause

The REVERSE-VIDEO clause specifies the switching of the foreground color and background color of a screen item.

Format

REVERSE-VIDEO

Syntax Rule

The REVERSE-VIDEO clause can be specified only in an elementary screen item.

General Rule

The REVERSE-VIDEO clause is applicable for both ACCEPT and DISPLAY statements. A screen item that specifies the REVERSE-VIDEO clause is displayed with foreground color and background color reversed.

5.5.19 SECURE Clause

The SECURE clause specifies non-display of an input item.

Format

SECURE

Syntax Rule

The SECURE clause can only be specified in an input item or group screen item.

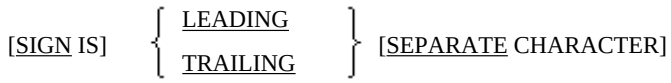
General Rules

1. If the SECURE clause is specified in a group screen item, it is applied to all of the group's subordinate input items.
2. When inputting data, via an ACCEPT statement, to a screen item specifying the SECURE clause, characters input to the screen item are not displayed on the screen, and the cursor remains at the first column of the screen item.

5.5.20 SIGN Clause

The SIGN clause specifies the position and representation mode of an operational sign of a screen item.

Format



Syntax Rule

The SIGN clause can be specified in elementary screen items or group screen items other than literal items. The syntax rules are the same as those of the SIGN clause in a data description entry. For details, see the topic titled "SIGN clause" in the "Data Description Entry" section earlier in this chapter.

General Rule

General rules are the same as those of the SIGN clause in a data description entry. For details, see the topic titled "SIGN clause" in the "Data Description Entry" section earlier in this chapter.

5.5.21 UNDERLINE Clause

The UNDERLINE clause specifies underlining in a screen item.

Format

UNDERLINE

Syntax Rule

The UNDERLINE clause can only be specified in an elementary screen item.

General Rule

The UNDERLINE clause is applicable for both ACCEPT and DISPLAY statements. A screen item specifying the UNDERLINE clause is displayed underlined.

5.5.22 USAGE Clause

The USAGE clause specifies the usage of a screen item.

Format

[USAGE IS] DISPLAY

Syntax Rules

1. The USAGE clause can be specified in elementary or group items other than literal items.
2. If the USAGE clause is specified in a group screen item, it is also applied to all subordinate screen items within the group. Elementary screen items whose USAGE IS DISPLAY is called a "screen item for display".

General Rules

1. The USAGE clause specifies the representation format of a data item on the screen. The USAGE IS DISPLAY clause indicates that the representation format of a data item is standard data format.
2. If the USAGE clause is not specified in a group screen item, the usage of all elementary items that depend on the group screen item is assumed to be display. Also, the usage of an elementary screen item that omitted the USAGE clause is assumed to be display.
3. Other general rules are the same as those of the USAGE clause in a data description entry. For details, see the section titled "USAGE clause."

5.5.23 VALUE Clause

The VALUE clause specifies the value of a literal item.

Format

VALUE IS literal-1

Syntax Rules

1. The VALUE clause can only be specified in a literal item. The VALUE clause and PICTURE clause cannot both be specified.
2. The literal-1 must be a nonnumeric literal, hexadecimal nonnumeric literal, or national nonnumeric literal.

General Rule

Literal-1 specifies the value of a literal item, that is, the value to be displayed on the screen.

5.6 Report Description Entry

A report description entry defines a report-name, the layout of each page of the report, and a control data item. A report description entry can only be defined in the REPORT SECTION.

Note

The report description entry cannot be used in [Winx64], [LinuxIPF] or [.NET].

Format

RD report-name-1

[CODE clause]

[CONTROL clause]

[PAGE clause].

Syntax Rules

1. The level indicator RD must be specified at the beginning of a report description entry, followed by report-name-1.

2. Report-name-1 can only be specified in the REPORT clause.
3. The clauses that follow report-name-1 can be specified in any order.
4. The qualifier for the report's LINE-COUNTER, PAGE-COUNTER, and the data-name defined in the REPORT SECTION is report-name-1.

PAGE-COUNTER, LINE-COUNTER Special Registers

For every report description entry defined, the special registers PAGE-COUNTER and LINE-COUNTER are automatically generated by the compiler. The special register PAGE-COUNTER is called the "page-counter" and the special register LINE-COUNTER is called the "line-counter."

Page Counter Rules

1. The PAGE-COUNTER can be specified in the following places:
    a. In a SOURCE clause of a report group description entry in the REPORT SECTION.
    b. Anywhere an integer value can be specified in the PROCEDURE DIVISION.
2. If two or more report description entries are defined in a program, when referencing the variable PAGE-COUNTER in a COBOL program, the following rules must be observed:
    a. When referencing PAGE-COUNTER in the PROCEDURE DIVISION, it must be qualified with report-name-1.
    b. PAGE-COUNTER can be specified in a SOURCE clause without qualification. If PAGE-COUNTER is not qualified, it is implicitly qualified with the report-name associated with the report group entry containing the SOURCE clause. If the PAGE-COUNTER of another report is referenced, it must be explicitly qualified with the report-name defined in the other report description entry.
3. The value of PAGE-COUNTER is updated by the report writer control system as follows:
    a. If an INITIATE statement is executed, PAGE-COUNTER is initialized to the value 1.
    b. The value of the PAGE-COUNTER is incremented by 1 whenever the report writer control system outputs a form feed (i.e. a new page).
4. Statements in the PROCEDURE DIVISION can update the PAGE-COUNTER value.

LINE-COUNTER Rules

1. The special register LINE-COUNTER can only be referenced in the following places:
    a. In the SOURCE clause of a report group description entry of the REPORT SECTION.
    b. Anywhere in the PROCEDURE DIVISION where an integer can be specified.
2. Statements in the PROCEDURE DIVISION cannot update the value of the special register LINE-COUNTER; it can only be updated by the report writer control system.
3. If two or more report description entry are defined in a program, when referencing the variable LINE-COUNTER in a COBOL program, the following rules must be observed:
    a. When referencing LINE-COUNTER in the PROCEDURE DIVISION, it must be qualified with report-name-1.
    b. LINE-COUNTER can be specified in a SOURCE clause without qualification. If LINE-COUNTER is not qualified, it is implicitly qualified with the report-name associated with the report group entry containing the SOURCE clause. If the LINE-COUNTER of another report is referenced, it must be explicitly qualified with the report-name defined in the other report description entry
4. The value of LINE-COUNTER is updated by the report writer control system as follows:
    a. If an INITIATE statement is executed, LINE-COUNTER is initialized to the value 0.
    b. The value of LINE-COUNTER is reset to zero whenever the report writer control system initiates a form feed (i.e. a new page).

5. When a report group processes a non-printable item, or processes a group whose printing is suppressed by a SUPPRESS statement, the value of LINE-COUNTER does not change.
6. The LINE-COUNTER value indicates the line number of the line printed. The value of LINE-COUNTER after the printing of a report group is determined by the presentation rules for each report group. For details of the presentation rules for a report group, see the topic titled "Report Group Presentation Rules" later in this section.

5.6.1 CODE Clause

The CODE clause specifies a literal that is appended to the print line of a report.

Format

CODE literal-1

Syntax Rules

1. Literal-1 must be a two-character nonnumeric literal.
2. If the CODE clause is specified for one report of a report description entry, it must be specified for all other reports, if any, of the same report description entry.

General Rules

1. If the CODE clause is specified, literal-1 is automatically appended to the first two characters of each print line of the report.
2. If more than one report is generated and output to a report file (verses reports output directly to a printing device), and if these reports are generated simultaneously, literal-1 can be used to identify which line belongs to which report, and the report file can be processed to separate the reports.
3. The CODE clause should not be used when the reports are output directly to a printing device.

5.6.2 CONTROL Clause

The CONTROL clause specifies the control fields and hierarchy of a report.

Format

```
{ CONTROL IS    } { {data-name-1} ...      }
{ CONTROLS ARE  } { FINAL [data-name-1]... }
```

Syntax Rule

1. Data-name-1 must not be defined in the REPORT SECTION.
2. If more than one data-name-1 is specified for a given report, the value of data-name-1 must not be repeated.
3. When defining the contents of the control field data-name-1, variable occurrence data items cannot be defined.

General Rules

1. Data-name-1 and FINAL specify the control hierarchy. FINAL is the highest-level control and is a control field for the entire report. The first specification of data-name-1 indicates the highest level control (after the FINAL control), the next occurrence of data-name-1 indicates the second highest level control, the third occurrence data-name-1 indicates the third level control, and so on.
2. The data item specified in data-name-1 is called a "control data item." When the value of a control data item changes, it is called a "control break".

   When a GENERATE statement is first executed first for a specific report, the report writer control system saves the values of all control data items related to the report. When subsequent GENERATE statements are executed for the

report, the report writer control system checks for changes in the values of control data items, beginning with the highest level control data item to the lowest control data item. If the value of a control data item changes, a control break occurs. When a control break occurs at a certain level, it also triggers all subordinate control breaks.

3. When a GENERATE statement is executed, the report writer control system checks, beginning with the highest level control data item through the lowest level control data item, for control breaks by comparing the value of the control data items with their previous values. The report writer control system performs one of the following comparison checks:

    a. If a control data item is a numeric data item, the check is a comparison of two numeric operands.

    b. If a control data item is an index data item, the check is a comparison of two index data items.

    c. If a control data item is National data item or national edited data item, the check is a comparison of two National operands.

    d. If a control data item is not (a), (b), or (c), the check is a comparison of two nonnumeric operands.

4. If there is no control data item corresponding to the most comprehensive control group in a report, FINAL is specified.

5.6.3 PAGE Clause

The PAGE clause specifies the page length and the various subdivisions within the page of a report group.

Format

```
PAGE [ LIMIT IS  ] integer-1 [ LINE  ]
     [ LIMITS ARE ]          [ LINES ]
  [HEADING integer-2] [FIRST DETAIL integer-3]
  [LAST DETAIL integer-4] [FOOTING integer-5]
```

Syntax Rules

1. The phrases HEADING (line number on the page for the page and report headings), FIRST DETAIL (line number on the page of the page body), LAST DETAIL (last line on the page of the page body), and FOOTING (line number on the page where a control footing can end) can be specified in any order.
2. Integer-1 may contain three significant digits.
3. Integer-2 must be greater than zero.
4. Integer-3 must be larger than integer-2.
5. Integer-4 must be larger than integer-3.
6. Integer-5 must be larger than integer-4.
7. Integer-1 must be larger than integer-5.
8. The following is the vertical page range for a report group specifying the PAGE clause.

    a. If a report heading report group is output to a single page that contains no other report groups (for example, a report cover sheet), it is output in the range of lines from integer-2 to integer-1. If a report heading report group is not output to a single page, it is output in the range of lines from integer-2 to the line above integer-3.

    b. If a page heading report group is specified, it is output in the range of lines from integer-2 to integer-3.

    c. If a control heading group or detail report group is defined, it is output in the range of lines from integer-3 to integer-4.

    d. If a control footing group is defined, it is output in the range of lines from integer-3 to integer-5.

    e. If a page footing report group is defined, it is output in the range of lines from the value of integer-5 plus 1 to integer-1.

f. If a report footing report group is displayed in one page, it is displayed in the range of lines from integer-2 to integer-1. If a report footing report group is not displayed in one page, it is displayed in the range of lines from integer-5 plus 1 to integer-1.

9. An entire report group must be defined to be output on one page. The report writer control system cannot output a multiple-line report group that continues onto another page.

General Rules

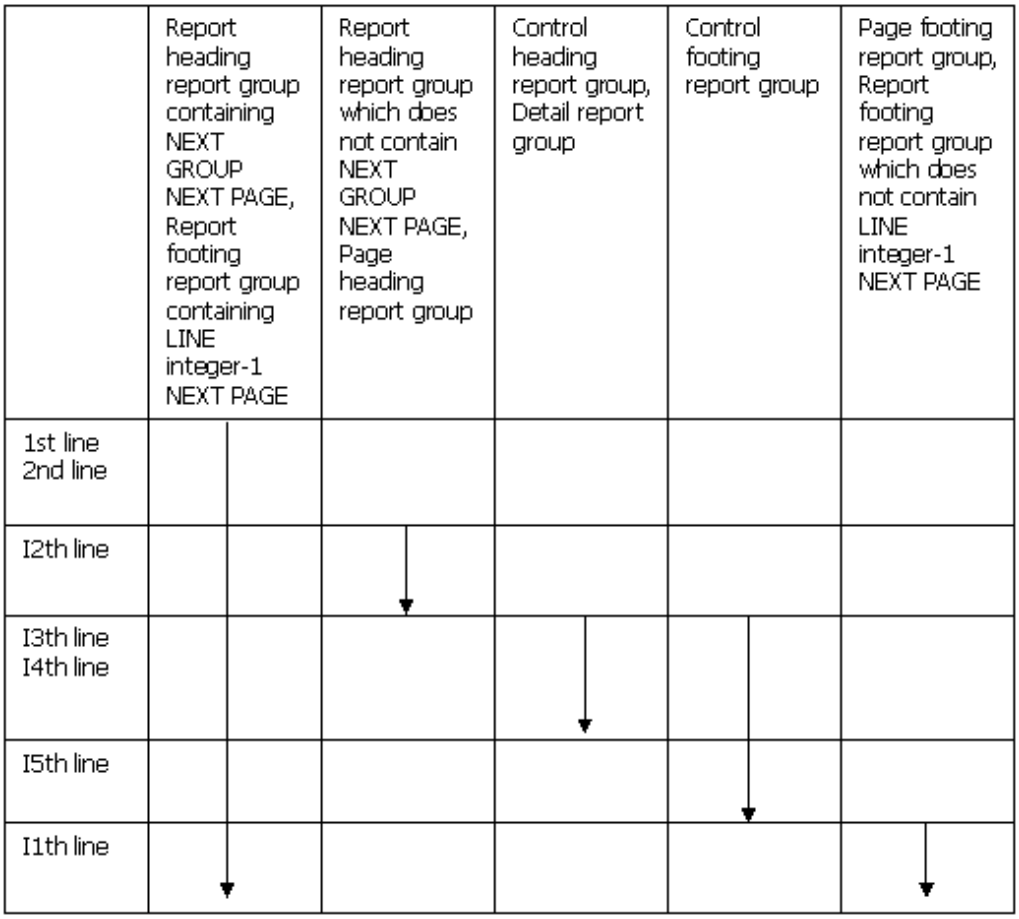
1. The vertical layout of a page of a report is determined by the integer values specified in the PAGE clause. Meaning of each integer is as follows:
   a. Integer-1 specifies number of lines of each page of a report.
   b. HEADING integer-2 specifies the first line number to display a report heading report group or page heading report group.
   c. FIRST DETAIL integer-3 specifies the first line number of the report body. A report heading report group (without the NEXT GROUP NEXT PAGE phrase) will not be output on or beyond the line number specified by integer-3.
   d. LAST DETAIL integer-4 specifies the last line number that a control heading group or detail report group can be printed on.
   e. FOOTING integer-5 specifies the last line number where a control footing group can end. A report footing report group (without LINE integer-1 NEXT PAGE phrase) and page footing group are output after the line number specified by integer-5.
2. If the PAGE clause is defined and not all PAGE phrases are specified, the following values are assumed:
   a. If a HEADING phrase is omitted, integer-2 assumes a value of 1.
   b. If a FIRST DETAIL phrase is omitted, integer-3 is assumed to be the same as integer-2.
   c. If the LAST DETAIL and FOOTING phrases are both omitted, integer-4 and integer-5 are assumed to be the same as integer-1.
   d. If a FOOTING phrase is specified and the LAST DETAIL phrase is omitted, integer-4 is assumed to be the same as integer-5.
   e. If a LAST DETAIL phrase is specified and the FOOTING phrase is omitted, integer-5 is assumed to be the same as integer-4.
3. If the PAGE clause is omitted, a report consists of one page of undetermined length.
4. For report group presentation rules, see the topic titled "Report Group Presentation Rules" later in this section.

Page Area

The following table lists page areas set by the PAGE clause.

Report Group Indicated in the Area	First Line Number in the Area	Last line Number in the Area
Report heading report group containing NEXT GROUP NEXT PAGE, Report footing report group containing LINE integer-1 NEXT PAGE	integer-2	integer-1
Report heading report group which does not contain NEXT GROUP NEXT PAGE, Page heading report group	integer-2	(integer-3) - 1
Control heading group, Detail report group	integer-3	integer-4
Control footing group	integer-3	integer-5
Page footing group, Report footing report group which does not contain LINE integer-1 NEXT PAGE	(integer-5) + 1	integer-1

The following figure shows the page area.



I1: Value of integer-1 of the PAGE clause

I2: Value of integer-2 of the HEADING phrase

I3: Value of integer-3 of the FIRST DETAIL phrase

I4: Value of integer-4 of the LAST DETAIL phrase

I5: Value of integer-5 of the FOOTING phrase

5.7 Report Group Description Entry

A report group description entry defines the characteristics of a report group and the characteristics of each item in the report group. A report group description entry can only be specified in the REPORT SECTION.

A report group description entry follows a report description entry. A complete report group description entry consists of three layers.

Note

The report group description entry cannot be used in [Winx64], [LinuxIPF], [Linux64] and [.NET].

Format 1

Report group description entry of the first layer

```
01 [data-name-1]
    [LINE NUMBER clause]
```

[NEXT GROUP clause]

TYPE clause

[USAGE clause].

Format 2

Report group description entry of the second layer

level-number [data-name-1]

[LINE NUMBER clause]

[USAGE clause].

Format 3

Report group description entry of the third layer

level-number [data-name-1]

[BLANK WHEN ZERO clause]

[COLUMN NUMBER clause]

[GROUP INDICATE clause]

[JUSTIFIED clause]

[LINE NUMBER clause]

PICTURE clause

[SIGN clause]

[USAGE clause]

{ SOURCE clause / SUM clause / VALUE clause } .

Syntax Rules

1. If data-name-1 is specified at the beginning of a report group description entry, data-name-1 must directly follow a level-number. All other clauses may be specified in any order.
2. Format 2 level-number of must be within the range of 02 to 48. Format 3 level-number of must be within the range of 02 to 49.
3. A report group description entry consists of three layers, layers 1, 2 and 3. The rules for applying these layers are shown below:
    a. The first entry of a report group description entry must be format 1 (layer 1).
    b. Report group description entries of format 2 (layer 2) or format 3 (layer 3) may follow a format 1 layer.
    c. If a report group description entry of format 2 is written, at least one format 3 report group description entry must follow.
    d. A report group description entry of format 3 must define an elementary item.
4. Data-name-1 of a report group description entry format 1 is only specified in the following cases:
    a. When a detail report group is referenced by a GENERATE statement
    b. When a detail report group is referenced by an UPON phrase of a SUM clause
    c. When a report group is referenced by a USER BEFORE REPORTING sentence.
    d. When a report group is referenced in the UPON phrase of a SUM clause.

5. A report group description entry of format 2 must specify at least one clause.
6. If a report group description entry of format 2 specifies data-name-1, data-name-1 can only used for qualifying a sum counter.
7. Specification of the USAGE clause in the REPORT SECTION is only for declaring the usage of printable data items.
    a. If a USAGE clause is specified in a format 3 entry, the entry must define printable data item.
    b. If a USAGE clause is specified in a format 1 or a format 2 entry, a printable data item must be defined in at least one of the subordinate entries.
8. If an entry specifies the LINE NUMBER clause, and if subordinate format entries are also specified, they cannot also specify a LINE NUMBER clause.
9. A data item defined with a format 3 report group description entry is called a "printable item". The following rules must be observed when using a format 3 report group description entry:
    a. The GROUP INDICATE clause can only be specified in a detail report group.
    b. The SUM clause can only be specified in a control footing group.
    c. If a format 3 report group description entry does not contain a LINE NUMBER clause, but does contain the COLUMN NUMBER clause, then the format 3 entry must be subordinate to a format 1 or a format 2 report group description entry containing a LINE NUMBER clause.
    d. If data-name-1 in a format 3 report group description entry is specified, it can be referenced only when the entry defines a sum counter.
    e. A COLUMN NUMBER clause must be defined if the entry also specifies a VALUE clause.
    f. Within all entries of a report group, the COLUMN NUMBER clause cannot both be specified.
10. The following five combinations of clauses are allowed in a format 3 entry :

Clause	Combination of Clauses				
	1	2	3	4	5
PICTURE clause	Necessary	Necessary	Necessary	Necessary	Necessary
COLUMN NUMBER clause	-	Necessary	Applicable	Applicable	Necessary
SOURCE clause	-	-	Necessary	Necessary	-
SUM clause	Necessary	Necessary	-	-	-
VALUE clause	-	-	-	-	Necessary
JUST clause	-	-	Applicable	-	Applicable
BLANK WHEN ZERO clause	-	Applicable	-	Applicable	-
GROUP INDICATE clause	-	-	Applicable	Applicable	Applicable
USAGE clause	-	Applicable	Applicable	Applicable	Applicable
SIGN clause	Applicable	Applicable	Applicable	Applicable	Applicable
LINE clause	Applicable	Applicable	Applicable	Applicable	Applicable

Necessary: The clause must always be written.

Applicable: The clause is optional.

-: The clause must not be written.

General Rules

1. Format 1 is a report description entry. A report group is defined by the report description entry and all its subordinate entries.

2. The BLANK WHEN ZERO, JUSTIFIED, and PICTURE clauses in a report group description entry have the same function and properties of the BLANK WHEN ZERO, JUSTIFIED, and PICTURE clauses in a data description entry, respectively. The rules for these clauses are defined the section titled "Data Description Entry" earlier in this chapter.
3. If two or more format 1 report group description entries are defined for a report, different areas are allocated for each report group description entry.

5.7.1 COLUMN NUMBER Clause

The COLUMN NUMBER clause specifies a column on a print line.

Format

COLUMN NUMBER IS integer-1

Syntax Rules

1. The COLUMN NUMBER clause can be specified only in an elementary item in a report group. If the COLUMN NUMBER clause is specified, the entry must also specify the LINE NUMBER clause, or the entry must be subordinate to an entry that specifies a LINE NUMBER clause.
2. Within one print line, printable items must be written from left to right sequentially. Columns of printable items in a line must not overlap.

General Rules

1. The COLUMN NUMBER clause indicates that a sum counter (that is defined with the SOURCE or VALUE clause), or a SUM clause is to be output on a print line. If the COLUMN NUMBER clause is not specified, the item is not output to the print line.
2. Integer-1 indicates the leftmost column of a printable item.
3. The report writer control system outputs spaces in all columns of a print line that are not occupied by a printable item.
4. The leftmost column of a print line is column 1.

5.7.2 GROUP INDICATE Clause

The GROUP INDICATE clause defines a detail report group that will print when a control break or form feed occurs.

Format

GROUP INDICATE

Syntax Rule

The GROUP INDICATE clause can only be specified in a detail report group that defines a printable item.

General Rules

1. The GROUP INDICATE clause specifies that the printable item is output only for the first detail line in a group, and when a control break or form feed occurs.
2. The GROUP INDICATE is valid only in a detail group that defines an elementary item that has a SOURCE or VALUE clause specified.
3. If the GROUP INDICATE clause is specified, the SOURCE or VALUE clauses are space filled, except in the following cases:
    a. When the first detail line in a group is encountered
    b. When a form feed is issued

c. When a control break occurs for the group

4. If the detail report group that specifies the GROUP INDICATE clause omits both the PAGE and CONTROL clauses, the SOURCE or VALUE clauses are space filled after an INITIATE statement is executed.

5.7.3 LINE NUMBER Clause

Format

```
LINE NUMBER IS { integer-1 [ON NEXT PAGE] }
               { PLUS integer-2           }
```

The LINE NUMBER clause specifies the line number on the page on which a report group begins.

Syntax Rules

1. Integer-1 and integer-2 can contain up to 3 significant digits. The values of integer-1 and integer-2 must not exceed the size of the page as defined in the PAGE clause.

   Integer-2 can be zero.

2. An entry that contains the LINE NUMBER clause cannot be specified in an entry that is subordinate to an entry that specifies a LINE NUMBER clause.

3. Within a report group description entry, an absolute LINE NUMBER clause must be specified before any relative LINE NUMBER clause can be written.

4. If a report group description entry specifies a series of absolute LINE NUMBER clauses, the values specified in integer-1 must be written in ascending order. They do not need to be consecutive.

5. If the PAGE clause is omitted, the report group description entry can only specify line numbers with the relative LINE NUMBER clause.

6. Within a report group description entry, the NEXT PAGE phrase can be specified only once. If the NEXT PAGE phrase is specified, it must be specified in the first LINE NUMBER clause of the group.

7. A LINE NUMBER clause with a NEXT PAGE phrase can only be specified in report group description entries that define a report group body or report group footings.

8. An entry that defines a printable item must contain the LINE NUMBER clause or be subordinate to an entry that contains the LINE NUMBER clause.

9. In a report group description entry that defines a page footing group, the first LINE NUMBER clause must use an absolute line number.

General Rules

1. The LINE NUMBER clause specifies the line number on the page on which a report group begins.

2. The report writer control system does the vertical positioning specified in the LINE NUMBER clause before the print line is output.

3. Integer-1 specifies an absolute line number. The absolute line number specifies the line number on the page where a print line is to be output.

4. Integer-2 specifies a relative line number. If the first LINE NUMBER clause of the group specifies an absolute line number and the next LINE NUMBER clause specifies a relative line number, the line number where a print line is output is calculated by adding the line number of the print line directly before the report group and integer-2. If integer-2 is zero, the print line is output on the same line as the previous print line.

   If the first LINE NUMBER clause of a report group specifies a relative line number, the line number where a print line is output is determined according to the topic titled "Report Group Presentation Rules" later in this section.

5. If a NEXT PAGE clause is specified, the report group is output on the line number specified by integer-1 on a new page.

5.7.4 NEXT GROUP Clause

The NEXT GROUP clause specifies where on the page the next group should be output.

Format

```
NEXT GROUP IS { integer-1
              | PLUS integer-2
              | NEXT PAGE }
```

Syntax Rules

1. A report group description entry specifying the NEXT GROUP clause must also specify the LINE NUMBER clause.
2. The values of integer-1 and integer-2 may contain up to 3 significant digits.
3. Within a report description entry that omits the PAGE clause and contains the NEXT GROUP clause, the NEXT GROUP clause must specify relative line numbers.
4. Within a report group description entry defining a page footing group, the NEXT PAGE phrase of the NEXT GROUP clause cannot be specified.
5. The NEXT GROUP clause cannot be specified in a report footing group or page heading group.

General Rules

1. The vertical positioning specified by the NEXT GROUP clause is performed after outputting the report group containing the clause.
2. The report writer control system calculates the new value of the current line by combining the information specified in the NEXT GROUP clause, the information specified in the TYPE and PAGE clauses, and the value of the current line.
3. If the NEXT PAGE clause is not specified in the highest control footing group, it is ignored.
4. The NEXT GROUP clause in a report body group affects the position where the next report body group is output. The NEXT GROUP clause in a report heading group affects the position where a page heading group is output. The NEXT GROUP clause in a page footing group affects the position where a report footing group is output.

5.7.5 SIGN Clause

The SIGN clause specifies the position of operational signs and the expression format.

Format

```
[SIGN IS] { LEADING  } SEPARATE CHARACTER
          { TRAILING }
```

Syntax Rules

1. The SIGN clause can only be specified in numeric data items that contain picture symbol "S".
2. A numeric data item specifying the SIGN clause must be used for explicit or implicit display.
3. If the SIGN clause is specified in a report description entry, the SEPARATE CHARACTER phrase must be specified.

General Rules

1. The SIGN clause specifies the position of a sign and the expression format of a numeric data item. The SIGN clause specified for a numeric data item will be applicable only to that data item. The picture symbol "S" indicates the presence of a sign, but not the expression form or position.

2. When a numeric data item contains the picture symbol "S" and omits the SIGN clause, SIGN IS TRAILING is assumed.
3. A report group description entry specifying the SIGN clause must also specify the SEPARATE CHARACTER phrase, and the signs are handled as follows:
    a. An operational sign will be appended at the left end character position when LEADING is specified or at the right end character position when TRAILING is specified. This character position is different from the numeric position.
    b. PICTURE symbol "S" is counted in the size of this item in the standard data format.
    c. The operational signs for positive and negative are "+" and "-", respectively. If the data does not contain "+" or "-" and the SEPARATE CHARACTER clause is specified, processing results are undefined.
4. A numeric data item with the picture symbol "S" is regarded as a numeric data item with a sign. When conversion is necessary for calculations or comparisons of data items specifying the SIGN clause, conversion is performed automatically by the COBOL runtime system.
5. The expression format of a data item with a SIGN clause is described in the topic titled "SIGN clause" and the topic titled "USAGE clause" in the "Data Description Entry" section earlier in this chapter.

5.7.6 SOURCE Clause

The SOURCE clause specifies a sending data item that provides values to be transferred to a printable item.

Format

SOURCE IS identifier-1

Syntax Rules

1. Identifier-1 can be defined in any section of the data division. However, only the following identifiers from the REPORT SECTION can be specified:
    - PAGE-COUNTER
    - LINE-COUNTER
    - A sum counter of the same report that specifies the SOURCE clause
2. Identifier-1 specifies a sending data item of an implicit MOVE statement to transfer data to a printable item. Identifier-1 must be defined in accordance with the rules for a sending item of a MOVE statement. See the topic titled "Standard Alignment Rules" of the "Concept of Data Description" section in Chapter 1, "General Rules" and the topic "Transcription and Movement of Data" in the "Nucleus" section of Chapter 2, "COBOL Modules" for transcriptions rules.
3. If reference modification is to be performed on identifier-1, the high-order-end-character-position and length of the reference modifier must be a data-name or literal.
4. If identifier-1 is subscripted, the subscript cannot specify the "+" or "-" operators.

General Rules

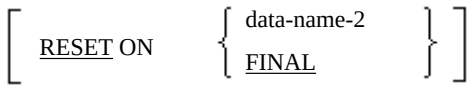
The report writer control system creates the printable line(s) for a report group immediately before outputting the report group. When the printable line is being created, an implicit MOVE statement is executed, where identifier-1 is the source operand and the printable item is the destination operand.

5.7.7 SUM Clause

The SUM clause creates a sum counter and specifies the names of data items to be summed.

Format

{SUM {identifier-1} ... [UPON {data-name-1} ...] } ...



Syntax Rules

1. The data item that is the subject of the report group description entry where the SUM clause appears must be defined as numeric or numeric edited. Identifier-1 must refer to a numeric data item. If identifier-1 is defined in the REPORT SECTION, it must refer to a sum counter.

   When an UPON phrase is omitted and the identifier in the SUM clause is a sum counter, this identifier must be defined in one of the following locations:

   a. in the same control footing group as that containing the SUM clause

   b. in a control footing group in a lower control hierarchy within the same report

   When the UPON phrase is specified, the identifier in the SUM clause must not be a sum counter.

2. Data-name-1 must be the name of a detail report group within the same report where the SUM clause is specified. data-name-1 can be qualified with a report name.

3. A SUM clause can be used only for a control footing group.

4. Data-name-2 must be one of the data-names specified in a CONTROL clause of the report. Data-name-2 must not be a control data name lower in the control hierarchy of the control footing group where a RESET phrase is specified. When FINAL is specified in the RESET phrase, FINAL must also be specified in the CONTROL clause in the report.

5. The highest level qualifier allowed for a sum counter is the report name.

General Rules

1. The SUM clause specifies a sum counter. A sum counter is a numeric data item with an operational sign generated by the compiler. The size of the sum counter and location of the decimal point are determined according to the category of the data item specified by the report group description entry where the SUM clause has been specified. The size and decimal point location are determined as follows:

   a. When the related data item is numeric, the size and decimal position of the sum counter will be the same as those of the related data item.

   b. When the related data item is a numeric edited data item, the size of the sum counter will be the same as the number of digits in the related data item, and the decimal point location the same as that of the related data item.

   c. When the related data item is an alphanumeric or alphanumeric edited data item, the size of the sum counter is the smaller of: the size of the related data item excluding the editing character, or 18 characters. The sum counter is an integer.

2. The value of the data item in identifier-1 is added to the sum counter at execution time by the report writer control system. This addition complies with the rules for of ADD statement.

3. If more than one SUM clause is specified in an elementary item description entry of a report, only one sum counter is set.

4. If a SUM clause is specified in a description entry of a printable item, the sum counter will be the source data item. The report writer control system transfers the sum counter value to the printable item in accordance with the rules of a MOVE statement.

5. A data-name can be specified in an elementary item description entry containing a SUM clause. If specified, the data-name is the sum counter name and not the name of the printable item.

6. The value of the sum counter can be modified by statement in the PROCEDURE DIVISION.

7. The report writer control system adds the value of the data item specified by identifier-1 to the sum counter when executing GENERATE and TERMINATE statements. There are three kinds of sum operations for a sum counter:

subtotaling, cross footing, and rolling forward. Subtotaling will be executed after processing a control break while executing a GENERATE statement, and before displaying a detail report group.

Cross footing and rolling forward will be performed while processing a control footing group.

8. If the UPON phrase is specified, subtotaling can be performed selectively for each repeated detail report group specified by identifier-1.
9. The report writer control system adds each value (repetition of identifier-1 in the SUM clause) to the sum counter in accordance with the following:
    a. If the addend is a sum counter value within the same control footing group, the summing operation is called "cross footing".

       Cross footing is performed when the control footing group is processed due to control break. The cross footing operation is performed in the order that the sum counters are defined in the control footing group.

       If one of the addends is a sum counter value defined in the data description entry where this SUM clause is specified, the value of the sum counter immediately before this sum operation is used for the sum operation.

    b. When the addend is a sum counter value in a lower level control footing group, the sum operation of the sum counter and the addend is called "rolling forward". This operation is performed when a lower level control footing group is processed due to a control break.

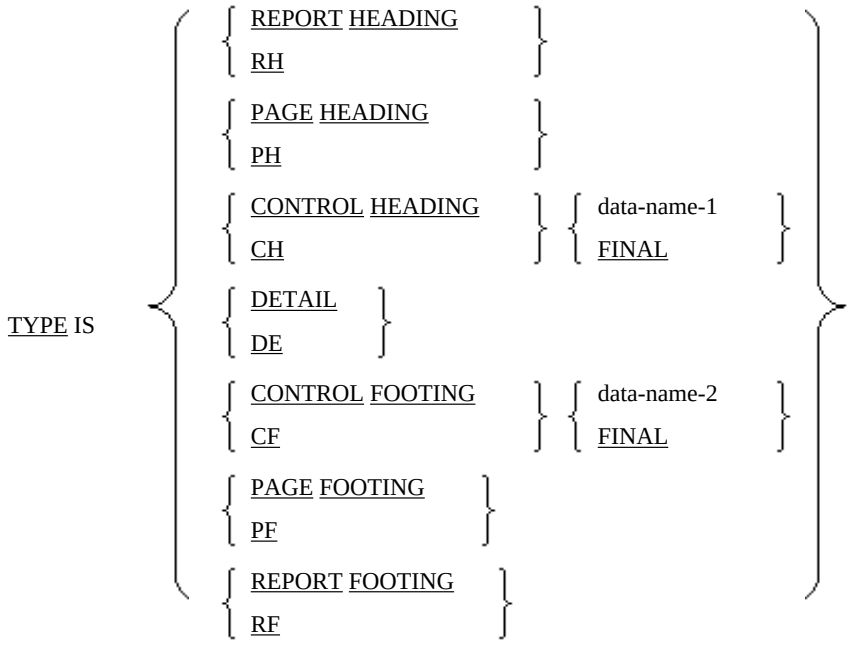
    c. When the addend is not a sum counter value, the addition of the addend and the sum counter is called "subtotaling." If an UPON phrase is specified in the SUM clause, the addend is added every time the GENERATE statement for a detail report group specified is executed. If an UPON phrase is not specified in a SUM clause, the addend is added every time a GENERATE statement specifying a report data-name is executed.
10. If the same identifier is specified as the addend two or more times, the value is added to the sum counter the number of times specified. If the same data-name is specified in an UPON phrase two or more times, the adding of data-name will be performed the number of times specified every time a GENERATE statement is executed for the detail report group that contains the SUM clause.
11. See the topic titled "GENERATE Statement (Report Writing)" in the "Statements" section of Chapter 6, the "Procedure Division" for the subtotaling to be performed when a GENERATE statement specifying a report name is executed.
12. When the RESET phrase is omitted then, when processing the control footing group containing the sum counter, the report writer control system sets the sum counter to zero. When the RESET phrase is specified, after processing the control footing group containing the sum counter, the report writer control system sets the sum counter to zero. For information on the timing of report group processing is discussed in the topic "Type Clause" later in this section.

    When an INITIATE statement is executed, the report writer control system sets all of the sum counters for the report to zero.

5.7.8 TYPE Clause

The TYPE clause specifies the type of the report group and the timing of its processing.

Format



Syntax Rules

1. The following terms are synonymous:
   - REPORT HEADING and RH
   - PAGE HEADING and PH
   - CONTROL HEADING and CH
   - DETAIL and DE
   - CONTROL FOOTING and CF
   - PAGE FOOTING and PF
   - REPORT FOOTING and RF
2. Only one report heading group, page heading group, control heading group with the FINAL phrase specified, control footing group with the FINAL phrase specified, page footing group, and report footing group can be specified in a report.
3. The page heading and the page footing groups can only be specified when the PAGE clause is specified in the associated report description entry.
4. Data-name-1, data-name-2 and FINAL must be specified in the CONTROL clause of the report description entry. For each data-name or FINAL specified in the CONTROL clause of the report description entry, only one control heading group and only one control footing group can be specified. However, both can be omitted.
5. The SOURCE clause of a page heading group, a SOURCE clause of a page footing group, or a related USE procedure must not contain a reference to a control data item.
6. A SOURCE clause or a related USE procedure for a control footing group, a page heading group, a page footing group, and a report footing group, must not contain a reference to the following data items:
   a. A group item containing a control data item
   b. A subordinate data item of a control data item
   c. A data item which is created by redefining or renaming a part or the all of a control data item
7. When the GENERATE statement specifies a report name, the associated report description entry cannot define more than one detail report group. Reports that are generated with GENERATE statements that do not specify a data-name can omit defining a detail report group.

8. A report description entry must contain at least one report body group description.

General Rules

1. Detail report groups are output when explicitly referenced by the GENERATE statement. All other report groups are automatically processed and output by the report writer control system.
2. The REPORT HEADING group of a report is processed only once and is the first report group processed by the report writer control system. The first report group processed in a report is called the "report heading group". The report heading group is processed when the first GENERATE statement for the report is executed.
3. Except as provided below, the PAGE HEADING phrase defines the first report group that is processed by the report writer control system for each page of the report. The first report group on a page of a report is called a "page heading group".
    a. When a page consists of only one report heading or report footing group, any page heading group defined is ignored
    b. If a report heading group is not printed on a page by itself, once output, the page heading report group is processed as the second report group
4. The CONTROL HEADING phrase specifies a report group that is processed by the report writer control system at the beginning of a named control group. The first report group in a control group is called a "control heading group".

    If FINAL is specified, it is executed during processing of the first GENERATE statement for the report. When the report writer control system detects a control break during execution of a GENERATE statement, all control heading groups in all levels below the highest level are processed.
5. The DETAIL phrase specifies a report group to be processed by the report writer control system while executing a GENERATE statement that specifies the detail report group name. A report group specifying the TYPE IS DETAIL clause is called a "detail report group". A detail report group, a control heading group, and a control footing group are collectively known as a "report body group."
6. The CONTROL FOOTING phrase defines a report group to be processed by the report writer control system at the end of a named control group. The last report group in a control group is called a "control footing group".

    If the FINAL phrase is specified, it is processed once in a report as the last main group of the report. When the report writer control system detects a control break during execution of a GENERATE statement, it processes all control footing groups in all levels below the highest level. If the GENERATE statement is executed at least once for the report, all the control footing groups are output during execution of the TERMINATE statement.
7. 7. The PAGE FOOTING phrase defines a report group to be processed by the report writer control system as the last report group of each page within the report except as provided below. The last report group on a page within a report is called a "page footing group".
    a. If a page only contains one report heading or report footing group, any PAGE FOOTING group defined is ignored
    b. If a report footing group is not printed on a page by itself, the page footing group is processed after the report footing group
8. The REPORT FOOTING phrase defines a report group that is processed once by the report writer control system as the last report group of the report. The last report group in the report is called a "report footing group". If a GENERATE statement is executed at least once for the report, all the control footing groups are output while executing the TERMINATE statement.
9. The report writer control system processes a report heading group, a page heading group, a control heading group, a page footing group, and a report footing group in the following order:
    a. If any USE BEFORE REPORTING procedure referring to a data-name in the report group exists, the USE procedure is executed.
    b. If a SUPPRESS statement is executed for the report group, the report group is not output.
    c. If a SUPPRESS statement is not executed, and the report group is printable, a print line is created and the report groups are output in compliance with the report group presentation rules.

10. The report writer control system processes the control footing groups in the following order:

    a. Calculates the totals of the sum counters cross-wise. Namely the sum counters defined in this report group and the operand of the SUM clause in the same report group are added to the main body sum counter in the SUM clause.

    b. The sum counters are summed by rolling forward. Namely, all the sum counters defined in this report group and the operands of the SUM clause in the higher level control footing group are added to the sum counter in that higher level control footing group.

    c. If any USE BEFORE REPORTING procedure that refers to a data-name in this report group exists, the USE procedure is executed.

    d. If a SUPPRESS statement is executed and the report group is not to be printed, process (f) will be executed.

    e. If a SUPPRESS statement is not executed or the report group is to be printed, printable lines are created and report groups are output in compliance with the control footing group presentation rules.

    f. In processing this level in the control hierarchy, the sum counter may be set to zero as required.

    When a control break occurs, the report writer control system creates and outputs control footing groups where control breaks were detected. This is done from the lowest level to the highest level in compliance with the GENERATE statement presentation rules. At this time, if no control footing group has been defined for a control data item, the report writer control system performs process (f) so long as the control data name has been specified in the RESET phrase in the report description.

11. The processes to be performed by the report writer control system for a detail report group corresponding to a GENERATE statement with a data-name specified will be one of the descriptions (a) through (e) provided below.

    When only one detail report group is defined in the report description, the processes to be performed by the report writer control system that corresponds to the GENERATE statement with a report name specified will be one of the descriptions (a) through (d). The report writer control system processes the report as if a GENERATE statement with a data-name specified had been executed.

    When no detail report group has been defined in a report description, the processes to be performed by the report writer control system for a corresponding GENERATE statement with a report name specified will be (a). The report writer control system processes the report as if a GENERATE statement with a data-name specified had been executed, where only one detail report group was defined in the report description.

    a. Executes all the subtotaling operations, as specified in the detail report group.

    b. If there is a USE BEFORE REPORTING procedure that references a data-name in this report group, the system executes the USE procedure.

    c. If the SUPPRESS statement has been executed, or if the report group is not to be printed, the system terminates the process for the report group.

    d. If the process for the detail report group depends on the GENERATE statement, the system terminates the process for the report group.

    e. If neither (c) nor (d), the report writer control system creates a printable line, and outputs the report group in compliance with the detail report group presentation rules.

12. While processing a control heading group, control footing group, or detail report group in compliance with general rules (9) and (10), the report writer control system may execute a page break. Further, the control system may execute the following page footing group and page heading report group processes before actually outputting the report body group. The report writer control system does this by suspending the output of the report body group after deciding to output these groups.

13. The values obtained by referencing a control data item are as shown below. The stored value of a control data item that the report writer control system uses to detect control breaks is called an "old value". See the topic titled "CONTROL clause" earlier in this section for information on control data items.

    a. The old value is used if a control data item is referenced in the associated USE procedure or in the associated SOURCE clause in processing a control break of a control footing group.

b. When a TERMINATE statement is executed, the report writer control system assumes that the highest level control break has been detected. The old value is used if a control data item is referenced in the SOURCE clause in the control footing group, report footing report group or in the associated USE procedure.

c. If a data item is referenced in the report group and in the associated USE procedure, the current value contained in the data item in processing the report group will be used, except cases (a) and (b).

5.7.9 USAGE Clause

The USAGE clause specifies the expression format for a data item in storage.

Format

[USAGE IS] DISPLAY

Syntax Rules

1. A USAGE clause can be specified in any data description entry.
2. When a USAGE clause is specified in a data description entry of a group item, a USAGE clause can be specified in subordinate elementary items or subordinate group items.
3. In a report group data description entry only USAGE IS DISPLAY can be specified.

General Rules

1. If a USAGE clause is specified for a group item, it will be applied to all the elementary items within the group.
2. A USAGE clause specifies the expression format of a data item in storage.
3. The USAGE IS DISPLAY clause indicates that the expression format of a data item in the standard data format.
4. A USAGE clause that is specified for an elementary item or a group item containing elementary items is regarded to be for display purposes.

5.7.10 VALUE Clause

The VALUE clause specifies a value for a printable item.

Format

VALUE IS literal-1

Syntax Rules

1. A signed numeric literal must correspond to the PICTURE character-string that describes the signed numeric data item.
2. A numeric literal in a VALUE clause must be a value within a range as defined by the PICTURE clause. It must not cause truncation of any numeric value other than 0. A nonnumeric literal in a VALUE clause must not exceed the size indicated in the PICTURE clause.

General Rules

1. The VALUE clause must not conflict with any other clauses specified in the data description entry for the item or the group item that contains the item. In addition, the following rules apply:
    a. When the category is numeric character, literal-1 in the VALUE clause must be a numeric literal.
    b. When the category is alphabetic character, alphanumeric character, alphanumeric edited, or numeric edited, literal-1 in the VALUE clause must be a nonnumeric literal. The literal is treated as though the data item's category is alphanumeric character.

F

c. If the category is National or national edited, literal-1 in the VALUE clause must be a national nonnumeric literal. This literal is treated as though the data item's category is national language.

d. If the category is Boolean, literal-1 in the VALUE clause must be a Boolean literal.

e. If any editing characters are included in the picture of the data item, they are ignored when initializing the value of the data item. Therefore, the literal in the data item should be specified in the edited form.

f. When initializing a data item any BLANK WHEN ZERO or JUSTIFIED clauses are ignored.

2. The specified value is given to a printable item every time a report group is displayed if the elementary entry containing the VALUE clause does not contain a GROUP INDICATE clause. If the entry does contain this clause, the specified value is output only when the conditions specified in the topic "GROUP INDICATE Clause" earlier in this section are satisfied.

5.8 Report Group Presentation Rules

In this section the following items are described.

- Possible combinations of LINE NUMBER and NEXT GROUP clauses for each type of report group
- Requirements to use LINE NUMBER and NEXT GROUP clauses
- Processes to be performed for the LINE NUMBER and NEXT GROUP clauses by the report writer control system

5.8.1 How to Use the Presentation Table Rules

Presentation table rules are provided below for the report heading group, the page heading group, the page footing group, the report footing group, the detail report group, the control heading group, and the control footing group.

The first and the second columns in the presentation table rules show all possible combinations of the LINE NUMBER and the NEXT GROUP clauses for a given report group type. Therefore, for one combination of the LINE NUMBER and NEXT GROUP clauses, read the row in the presentation rule table from left to right.

Reading Applicable Rule Column

The applicable rule column in the presentation table rules is divided into two parts. The first part is applicable to report descriptions containing a PAGE clause, while the second part is applicable to those not containing a PAGE clause. A hyphen (-) indicates that no rule is applicable. The names and purposes of the rules in the applicable rule column are as provided below:

Upper Limit and Lower Limit Rules

These rules specify the vertical range within a page where the specified report group is output. If there is no PAGE clause, the report to be printed will not be split vertically. Therefore, no upper limit or lower limit rule for a report description without a PAGE clause appears in the table.

Page Break Rules

The page break rule is applicable to the report body group. Thus, the page break rule only appears in the presentation table rules for the report body group. The report writer control system determines whether the specified report body group can be output on the current page of the report at execution time by applying the page break rules. If a PAGE clause is not specified in a report description entry, the page break rules are not applicable to the report body group.

Starting Print Line Position Rules

The starting print line position rules specify where the report writer control system should output the first line of a report group on a report medium. The presentation table rule does not specify where the report writer control system output the second and subsequent print lines (if any) on the report medium. This is defined by the general rules for the LINE NUMBER clause.

Next Report Group Rules

The next report group rules specify how to use the NEXT GROUP clause.

LINE-COUNTER Final Value Setting Rules

The LINE-COUNTER final value setting rules specify the final value to be set for the LINE-COUNTER by the report writer control system after outputting the report group.

LINE NUMBER Clause Notation

The first column of the presentation table rules indicates the order of the LINE NUMBER clause within one report group. The symbols are defined below:

- "Absolute" means that a LINE NUMBER clause without a NEXT PAGE phrase is to be specified. In the LINE NUMBER clause without a NEXT PAGE phrase, specify an absolute line number. LINE NUMBER clauses without a NEXT PAGE phrase can be specified consecutively. After this, LINE NUMBER clauses with a PLUS phrase can also be specified consecutively.
- "PLUS" indicates that a LINE NUMBER clause with the PLUS phrase is to be specified. A LINE NUMBER clause with a PLUS phrase specifies a relative line position. Consecutive LINE NUMBER clauses with the PLUS specified can be written.
- "NEXT PAGE" indicates that a LINE NUMBER clause with a NEXT PAGE phrase is to be specified. A LINE NUMBER clause with NEXT PAGE specifies an absolute line position after changing the page. Following this clause, consecutive LINE NUMBER clauses without NEXT PAGE or LINE NUMBER clauses with PLUS can be written.
- "Omit" means omit the LINE NUMBER clause.

NEXT GROUP Clause Notation

The second column of the presentation table rules shows how to write a NEXT GROUP clause in one report group. Only one NEXT GROUP clause can be written in one report group. The meaning of the symbols is as shown below.

- "Absolute" indicates that a NEXT GROUP clause with integer-1 is to be specified.
- "PLUS" indicates that a NEXT GROUP clause with PLUS is to be specified.
- "NEXT PAGE" indicates that a NEXT GROUP clause with NEXT PAGE is to be specified.
- "Omit" indicates that a NEXT GROUP clause is to be omitted.
- "Prohibited" indicates that a NEXT GROUP clause cannot be specified.

Integer Saving Item for Next Report Group

The integer saving item for next report group is a data item that can be processed only by the report writer control system. If an absolute line number is specified in a NEXT GROUP clause, and the line number cannot be applied to the current page, the report writer control system saves the value to the integer saving item for next report group.

After performing the page break process, the report writer control system positions the next report body group using the value in the integer saving item for next report group.

5.8.2 Report Heading Group Presentation Rules

All the possible combinations of LINE NUMBER and NEXT GROUP clauses for the report heading group and their presentation rules are shown in the table below.

Combination of LINE NUMBER Clause and NEXT GROUP Clause		Applicable Rule						
		With a PAGE Clause					Without a PAGE Clause	
LINE NUMBER clause	NEXT GROUP clause	Upper limit	Lower limit	Printing starting line position	Next report group	Line counter final value setting	Printing starting line position	Line counter final value setting
Absolute	**Absolute**	(1)	(2) (a)	(3) (a)	(4) (a)	(5) (a)	Invalid combination	

Combination of LINE NUMBER Clause and NEXT GROUP Clause		Applicable Rule						
		With a PAGE Clause					Without a PAGE Clause	
LINE NUMBER clause	NEXT GROUP clause	Upper limit	Lower limit	Printing starting line position	Next report group	Line counter final value setting	Printing starting line position	Line counter final value setting
Absolute	**PLUS**	(1)	(2) (a)	(3) (a)	(4) (b)	(5) (b)	Invalid combination	
Absolute	**NEXT PAGE**	(1)	(2) (b)	(3) (a)	(4) (c)	(5) (c)	Invalid combination	
Absolute	**Omit**	(1)	(2) (a)	(3) (a)	-	(5) (d)	Invalid combination	
PLUS	**Absolute**	(1)	(2) (a)	(3) (b)	(4) (a)	(5) (a)	Invalid combination	
PLUS	**PLUS**	(1)	(2) (a)	(3) (b)	(4) (b)	(5) (b)	(3) (d)	(5) (b)
PLUS	**NEXT PAGE**	(1)	(2) (b)	(3) (b)	(4) (c)	(5) (c)	Invalid combination	
PLUS	**Omit**	(1)	(2) (a)	(3) (b)	-	(5) (d)	(3) (d)	(5) (d)
Omit	**Prohibited**	-	-	(3) (c)	-	(5) (e)	(3) (c)	(5) (e)

See the topic titled "How to Use the Presentation Tables Rules" earlier in this section to understand the table. Numbers in parentheses in the above table are explained below:

(1) Upper limit rule

The first line number where a report heading group can be output is to be specified with the HEADING phrase of the PAGE clause.

(2) Lower limit rule

a. The last line position where a report heading group can be output will be the value obtained by subtracting one from integer-3 as specified in FIRST DETAIL phrase of the PAGE clause.

b. The last line position where a report heading group can be output will be specified with integer-1 in the PAGE clause.

(3) Starting Print line position rule

a. The first printable line for a report heading group is output at the line number specified by the integer in the PAGE clause.

b. The first printable line for a report heading group is output at the line number indicated by the value obtained by adding the integer in the first LINE NUMBER clause and the value (integer-2 - 1) in the HEADING phrase of the PAGE clause.

c. Any report heading group is not output.

d. The first printable line in a report heading group is output at the line number obtained by adding the integer in the first LINE NUMBER clause to the LINE-COUNTER value (in this case zero).

(4) Next report group rule

a. The integer in the NEXT GROUP phrase must be greater than the line number used to output the last printable line of a report heading group. Also, it must be less than the line number indicated by integer-3 in the FIRST DETAIL phrase in the PAGE clause.

b. The value added to the integer in the NEXT GROUP phrase and the line number where the last printable line of a report heading group is to be output must be less than integer-3 in the FIRST DETAIL phrase of a PAGE clause.

c. The NEXT PAGE phrase of NEXT GROUP clause indicates that a single report heading group is output on the first page of the report. The report writer control system does not process other report groups while the report is positioned at the first page.

(5) LINE-COUNTER final value rule

a. The report writer control system moves the integer in the NEXT GROUP clause to LINE-COUNTER as the final value.

b. The report writer control system computes the final value by adding the integer in the NEXT GROUP clause and the line number of the last printable line of the report heading group output to LINE-COUNTER as the final value.

c. The report writer control system sets LINE-COUNTER to zero.

d. The last printable line number of a report heading group that is output is moved to LINE-COUNTER as the final value.

e. LINE-COUNTER will not be affected by any processing of a report group that does not contain printable items.

5.8.3 Page Heading Group Presentation Rules

All the possible combinations of LINE NUMBER and NEXT GROUP clauses for a page heading group and their presentation rules are shown in the table below.

Combination of LINE NUMBER Clause and NEXT GROUP Clause		Applicable Rule				
		With a PAGE Clause (*1)				
LINE NUMBER clause	NEXT GROUP clause	Upper limit	Lower limit	Printing starting line position	Next report group	Line-counter final value setting
Absolute	**Prohibited**	(1)	(2)	(3) (a)	-	(4) (a)
PLUS	**Prohibited**	(1)	(2)	(3) (b)	-	(4) (a)
Omit	**Prohibited**	-	-	(3) (c)	-	(4) (b)

*1 : Do not define any page heading group if any PAGE clause is not contained in a report description entry.

See the topic titled "How to Use the Presentation Table Rules" earlier in this section to understand the table. Numbers in parentheses in the above table are explained below:

(1) Upper limit rule

If a report heading group has been output on a page where a page heading report group is also to be output, the first line number where a page heading group can be output is the value of the last line number output for the report heading group plus one.
Otherwise, the first line number where a page heading group can be output is specified in the HEADING phrase of the PAGE clause.

(2) Lower limit rule

The last line number where a page heading group can be output is computed by subtracting one from integer-3 as specified in the FIRST DETAIL phrase of the PAGE clause.

(3) Printing starting line position rule

a. The first printable line for a page heading group is the line number specified by the integer in the LINE NUMBER clause for the group.

b. If a report heading group has already been output on the page where a page heading group is to be output, the first printable line for the page heading group is computed by adding the last printed line number of the report heading group to the integer specified in the first LINE NUMBER clause of the page heading.
Otherwise, the first printable line of a page heading group is computed by adding the value of the integer in the first LINE NUMBER clause in the page heading group to the value obtained by subtracting one from integer-2 in the HEADING phrase in a PAGE clause.

c. No page heading group is displayed.

(4) LINE-COUNTER final value setting rule

a. The report writer control system moves the line number of the last printable line of a page heading group to LINE-COUNTER as the final value.

b. LINE-COUNTER will not be affected by any processing of a report group that does not contain printable items.

5.8.4 Report Body Group Presentation Rules

All the possible combinations of LINE NUMBER and NEXT GROUP clauses for the control heading group, the detail report group and the control footing group, and their presentation rules are shown in the table below.

Combination of LINE NUMBER clause and NEXT GROUP clause		Applicable rule							
		With a PAGE clause						Without a PAGE clause	
LINE NUMBER clause	NEXT GROUP clause	Upper limit	Lower limit	Page break	Printing starting line position	Next report group	Line counter final value setting	Printing starting line position	Line counter final value setting
Absolute	**Absolute**	(1)	(2)	(3) (a)	(4) (a)	(5)	(6) (a)	Invalid combination	
Absolute	**PLUS**	(1)	(2)	(3) (a)	(4) (a)	-	(6) (b)	Invalid combination	
Absolute	**NEXT PAGE**	(1)	(2)	(3) (a)	(4) (a)	-	(6) (c)	Invalid combination	
Absolute	**Omit**	(1)	(2)	(3) (a)	(4) (a)	-	(6) (d)	Invalid combination	
PLUS	**Absolute**	(1)	(2)	(3) (b)	(4) (b)	(5)	(6) (a)	Invalid combination	
PLUS	**PLUS**	(1)	(2)	(3) (b)	(4) (b)	-	(6) (b)	(4) (d)	(6) (f)
PLUS	**NEXT PAGE**	(1)	(2)	(3) (b)	(4) (b)	-	(6) (c)	Invalid combination	
PLUS	**Omit**	(1)	(2)	(3) (b)	(4) (b)	-	(6) (d)	(4) (d)	(6) (d)
NEXT PAGE	**Absolute**	(1)	(2)	(3) (c)	(4) (a)	(5)	(6) (a)	Invalid combination	
NEXT PAGE	**PLUS**	(1)	(2)	(3) (c)	(4) (a)	-	(6) (b)	Invalid combination	
NEXT PAGE	**NEXT PAGE**	(1)	(2)	(3) (c)	(4) (a)	-	(6) (c)	Invalid combination	
NEXT PAGE	**Omit**	(1)	(2)	(3) (c)	(4) (a)	-	(6) (d)	Invalid combination	
Omit	**Prohibited**	-	-	-	(4) (c)	-	(6) (e)	(4) (c)	(6) (e)

See the topic titled "How to Use the Presentation Table Rules" earlier in this section to understand the table. Numbers in parentheses in the above table are explained below:

(1) Upper limit rule

The first line number where a report body group can be output is specified in the FIRST DETAIL phrase of the PAGE clause.

(2) Lower limit rule

The last line number where a control heading or a detail report group can be output is specified in the LAST DETAIL phrase in the PAGE clause.
The last line number where a control footing group can be output is specified in the FOOTING phrase of the PAGE clause.

(3) Page break rule

a. If the value of LINE-COUNTER is less than the integer in the first absolute LINE NUMBER clause, the report body group is output on the current page.
Otherwise, the report writer control system performs the page break process. If a page heading group has been defined, the report writer control system checks whether the integer saving item of the next report group has a value (when the last report body group of the previous page was output See (6)(a)). If the integer saving item has no value, the report body group is output on the current page. If the integer saving item has a value, it is transferred to LINE-COUNTER, zero is moved to the integer saving item for next report group, and rule (3)(a) is applied again.

b. If the report body group has already been output on the current page of the report, the report writer control system calculates a total by adding the contents of LINE-COUNTER and the integers of all the LINE NUMBER clauses for the report group. If the total is less than or equal to the integer of the lower limit for the report body group, the report group is output on the current page.
If this total exceeds the integer of the lower limit for the report body group, a page break process performed. If a page heading report group is defined, the report writer control system applies rule (3)(b) again.
If the body group has not been output on the current page of the report, the report writer control system checks whether the integer saving item for next report group (when the last report body group was output on the previous page See (6)(a)) has a value.
If the integer saving item has no value, the report body group is output on the current page of the report. Otherwise, the report writer control system transfers the value of the integer saving item to LINE-COUNTER, moves zero to the integer saving item for next report group, and calculates a testing total.
This total is calculated by adding LINE-COUNTER, 1 and the integers in all the LINE NUMBER clauses (excluding the first one) in the report body group. If this total does not exceed the lower limit integer for the report body group, the report body group is output on the current page. If this total exceeds the lower limit integer for the report body group, the report writer control system performs the page break process. If any page heading group has been defined, it outputs the report body group on the page after the process.

c. If the report body group has been output on the current page of the report, the report writer control system performs the page break process. If a page heading group has been defined, it applies (3)(c) after the process. If a report body group has not been output on the current page of the report, the report writer control system checks whether the integer saving item of the next report group (when the last report body group of the previous page was output) has a value (after the page break process see (6)(a)). If the integer saving item has no value, the report body group is output on the current page. Otherwise, the report writer control system transfers the integer saving value to LINE-COUNTER, and moves zero to the integer saving item for next report group. Then, if the value in LINE-COUNTER is less than the integer in the first mandatory LINE NUMBER clause, the report body group is output on the current page. Otherwise, the report writer control system performs the page break process. If a page heading group has been defined, the system outputs the report body group on the page after the process.

(4) Printing starting line position rule

a. The first printable line in the report body group is output at the line number specified by the integer in the LINE NUMBER clause for the group.

b. If the value in LINE-COUNTER exceeds the line number specified in the FIRST DETAIL phrase of the PAGE clause, and no report body group has been output on the current page, the first printable line for the current report body group will be output on the next line after the line indicated by the value of LINE-COUNTER.
If the value of LINE-COUNTER is greater than the line number specified in the FIRST DETAIL phrase of the PAGE clause, and the report body group has already been output on the current page, the first printable line of the current report body group will be output on the line indicated by the value computed by adding the LINE-COUNTER and the integer in the first LINE NUMBER clause for the current report body group. If the value in LINE-COUNTER is less than the line number specified by the FIRST DETAIL phrase of the PAGE clause, the first printable line of the current report body group will be output on the line specified in the FIRST DETAIL phrase.

c. The report body group will not be displayed.

d. The line number to output to be computed by adding the value of LINE-COUNTER and the integer in the first LINE NUMBER clause.

(5) Next report group rule

The integer in an absolute NEXT GROUP clause shall be larger than the line number specified by the FIRST DETAIL phrase in a PAGE clause, and be less than the line number specified by the FOOTING phrase in the PAGE clause.

(6) LINE-COUNTER final value setting rule

a. When a control footing group is output and if it is not the highest level among the control footing groups corresponding to the control break, the final value of LINE-COUNTER will be the line number which output the last printable line of the control footing group.
In all other cases, the report writer control system compares the line number of the last printable line of the report body group output with the integer specified in the NEXT GROUP phrase. If the former is less than the latter, the report writer control system moves the integer specified by the NEXT GROUP phrase to LINE-COUNTER as the final value. If the former is greater than the latter, the report writer control system moves the line number specified in the FOOTING phrase of the PAGE clause to LINE-COUNTER as the final value, and it moves the integer in the NEXT GROUP clause to the integer for next report group saving item.

b. When a control footing group is output, if it is not the highest level among the control footing groups corresponding to the control break, the final value of LINE-COUNTER is the line number which output the last printable line of the control footing group.
In all other cases, the report writer control system adds the integer in the NEXT GROUP clause to the line number of the last printable line of the report body group output, calculating a testing total. If the total is less than the line number specified in the FOOTING phrase of the PAGE clause, the report writer control system moves the total to LINE-COUNTER as the final value. If the total is greater than the line number specified in the FOOTING phrase of the PAGE clause, the report writer control system moves the line number specified in the FOOTING phrase of the PAGE clause to LINE-COUNTER as the final value.

c. When a control footing group is output and it is not the highest level among the control footing groups corresponding to the control break, the final value of LINE-COUNTER will be the line number of the last printable line of the control footing group output.
In all other cases, the report writer control system moves the line number specified in the FOOTING phrase of the PAGE clause to LINE-COUNTER as the final value.

d. The final value of LINE-COUNTER is the line number of the last printable line of the body group output.

e. Any process of the report body group that does not contain a printable item will not affect LINE-COUNTER.

f. When a control footing is output and it is not the highest level among the control footing groups corresponding to the control break, the final value of LINE-COUNTER will be the line number of the last printable line of the control footing group output.
In all other cases, the report writer control system adds the value of the line number where the last printable line was output and the integer in the NEXT GROUP clause and moves the total to LINE-COUNTER as the final value.

5.8.5 Page Footing Group Presentation Rules

All the possible combinations of LINE NUMBER and NEXT GROUP clauses for a page footing group and their presentation rules are shown in the following table.

Combination of LINE NUMBER clause and NEXT GROUP clause		Applicable rule With a PAGE clause (*1)				
LINE NUMBER clause	NEXT GROUP clause	Upper limit	Lower limit	Printing starting line number	Next report group	Line-counter final value setting
Absolute	**Absolute**	(1)	(2)	(3) (a)	(4) (a)	(5) (a)

Combination of LINE NUMBER clause and NEXT GROUP clause		Applicable rule With a PAGE clause (*1)				
LINE NUMBER clause	NEXT GROUP clause	Upper limit	Lower limit	Printing starting line number	Next report group	Line-counter final value setting
Absolute	**PLUS**	(1)	(2)	(3) (a)	(4) (b)	(5) (b)
Absolute	**Omit**	(1)	(2)	(3) (a)	-	(5) (c)
Omit	**Prohibited**	-	-	(3) (b)	-	(5) (d)

*1 Do not define any page footing group if any PAGE clause is not contained in a report description entry.

See the topic titled "How to Use the Presentation Table Rules" to understand the table. The numbers in parentheses in the above table are explained below:

(1) Upper limit rule

The first line number where a page footing group can be output is the sum of integer-5 in the FOOTING phrase of the PAGE clause plus 1.

(2) Lower limit rule

The last line number where a page footing group can be output is the value specified by integer-1 of the PAGE clause.

(3) Printing start line number rule

a. The first printable line of a page footing group is at the line number specified by the integer in the LINE NUMBER clause for the group.

b. Any page footing group is not displayed.

(4) Next report group rule

a. The integer in the NEXT GROUP clause must be greater than the line number where the last printable line of the page footing group was output. Also, it must be less than the line number specified by integer-1 in the PAGE clause.

b. The sum of the integer in the NEXT GROUP clause and the line number where the last printable line of the page footing group is output must be less than the line number specified by integer-1 in the PAGE clause.

(5) LINE-COUNTER final value setting rule

a. The report writer control system moves the integer in the NEXT GROUP clause to LINE-COUNTER as the final value.

b. The report writer control system moves the sum of the integer in the NEXT GROUP clause and the line number of the last printable line of the page footing group output to LINE-COUNTER as the final value.

c. The report writer control system moves the line number of the last printable line of the page footing group output to LINE-COUNTER as the final value.

d. LINE-COUNTER will not be affected by a process of a report group that does not contain a printable item.

5.8.6 Report Footing Group Presentation Rule

All the possible combinations of LINE NUMBER and NEXT GROUP clauses for the report footing group, and their presentation rules are shown in the table below.

Combination of LINE NUMBER clause and NEXT GROUP clause		Applicable rule						
		With a PAGE clause					Without a PAGE clause	
LINE NUMBER clause	NEXT GROUP clause	Upper limit	Lower limit	Printing starting line position	Next report group	Line counter final value setting	Printing starting line position	Line counter final value setting
Absolute	**Prohibited**	(1) (a)	(2)	(3) (a)	-	(4) (a)	Invalid combination	
PLUS	**Prohibited**	(1) (a)	(2)	(3) (b)	-	(4) (a)	(3) (d)	(4) (a)
NEXT PAGE	**Prohibited**	(1) (b)	(2)	(3) (c)	-	(4) (a)	Invalid combination	
Omit	**Prohibited**	-	-	(3) (e)	-	(4) (b)	(3) (e)	(4) (b)

See the topic titled "How to Use the Presentation Table Rules" to understand the table. Numbers in parentheses in the above table are explained below:

(1) Upper limit rule

a. If a page footing group has been output on the current page of the report, the first line number where a report footing group can be output is sum of the final value of LINE-COUNTER as set for the page footing group plus one.

b. In other cases, the first line number where a report footing group can be output will be the sum of integer-5 in the PAGE clause plus one.

c. The first line number where a report footing group can be output is specified by the HEADING phrase of the PAGE clause.

(2) Lower limit rule

The last line number where a report footing group can be output is specified by integer-1 in the PAGE clause.

(3) Printing start line position rule

a. The first printable line of a report footing group is the line number specified by the integer in the LINE NUMBER clause for the group.

b. If a page footing group has already been output on the current page of the report, the first printable line number of the report footing group is the sum of the final value of LINE-COUNTER as set by the page footing group and the integer in the first LINE NUMBER clause for the report footing group. If a page footing group has not been output, the first printable line number of the report footing group is the sum of the integer in the first LINE NUMBER clause for the report footing group and integer-5 in the FOOTING phrase in a PAGE.

c. If a NEXT PAGE phrase is specified in a LINE NUMBER clause, the single report footing group output on one page. The first printable line of a report footing group is the line number specified by the integer in the LINE NUMBER clause for the group.

d. The line number of the first printable line is the sum of LINE-COUNTER and the integer in the LINE NUMBER clause.

e. No report footing group will be displayed.

(4) LINE-COUNTER final value setting rule

a. The report writer control system moves the line number of the last printable line of a report footing group output to LINE-COUNTER as the final value.

b. LINE-COUNTER will not be affected by processing a report group that does not contain a printable item.

Chapter 6 Procedure Division

The procedure division contains the procedures to be executed by the object program. The procedure division is defined after the data division. The procedure division can be omitted if no object program is required.

6.1 Composition of the Procedure Division

The procedure division consists of declaratives and procedures.

Declaratives are specified in a block of code that begins with the keyword DECLARATIVES and ends with the keyword END DECLARATIVES.

A procedure consists of one paragraph, a set of several consecutive paragraphs, one section, or a set of several consecutive sections. A section contains all paragraphs belonging to the section.

Format 1

If the declarative is omitted (1):

```
PROCEDURE DIVISION.
{section-name SECTION.
[paragraph-name.
        [sentence ] ... ] ... } ...
```

Procedure

Format 2

If the declarative is omitted (2):

```
PROCEDURE DIVISION.
[paragraph-name.
        [sentence ] ... ] ...
```

Procedure

Format 3

If declaratives are included:

```
PROCEDURE DIVISION.
DECLARATIVES.
{section-name SECTION.
    USE statement. (*1)
[paragraph-name.
    [sentence ] ... ] ... } ...
END DECLARATIVES.
{section-name SECTION.
[paragraph-name.
    [sentence ] ... ] ... } ...
```

Declarative

Procedure

*1: A USE statement also includes a USE BEFORE REPORTING statement.

F

In this implementation of COBOL, if all of the procedure division statements are sequentially executed or in-line (using the xxx/END-xxx structure), the names of sections and paragraphs in the PROCEDURE DIVISION may be omitted.

Section

A section consists of a section header ("section-name SECTION") and any number of paragraphs. A section name must be followed by a period (separator). Paragraphs can be omitted. A section begins at a section header and ends at one of the following locations:

- Immediately before the next section
- When a section is a declarative, at END DECLARATIVES keyword
- At the end of the procedure division

Note that sentences and paragraphs are allowed before the section header, and sentences are allowed before paragraphs.

Paragraph

A paragraph consists of a paragraph name and any number of sentences. A paragraph name must be followed by a period (separator). A sentence can be omitted. A paragraph begins at the paragraph name and ends at one of the following locations:

- Immediately before the next paragraph name
- Immediately before the next section-name
- At the end of the procedure division
- When the paragraph is a declarative, at the END DECLARATIVES keyword

Statements

A sentence consists of any number of statements, and ends with a period (separator). The statements in a procedure division are executed in the order in which they are written.

Statements include conditional, imperative, and compiler-directing statements.

A conditional statement determines the next operation based on the truth value of the condition.

An imperative statement indicates a specific unconditional action to be taken by the object program. Several consecutive imperative statements can be written. A separator can be placed between imperative statements. Where a syntax diagram indicates "imperative-statement-n", several consecutive imperative statements without a separator period can be written.

A compiler-directing statement specifies an operation to be performed during compilation. These statements have no meaning during program execution. Compiler-directing statements include the COPY, REPLACE, and USE statements. The COPY and REPLACE statements can be written anywhere in the program source code. Chapter 7, "Source Text Manipulation" explains the COPY and REPLACE statements.

A conditional statement specifies statements to be executed depending on the truth value of a condition. A conditional statement can be changed to an imperative statement by putting an explicit scope terminator at the end of the statement. An explicit scope terminator is a word in form of "END-verb". For example, an ADD statement containing ON SIZE ERROR, but not containing an END-ADD phrase is a conditional statement, and that containing END-ADD is an imperative statement.

The following table lists conditional and imperative statements:

Statement	Difference Between Conditional and Imperative Statements
ACCEPT statement	Conditional statement (*1) if it contains ON EXCEPTION phrase or NOT ON EXCEPTION phrase.
	Imperative statement otherwise.
ADD statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
ALTER statement	Imperative statement
CALL statement	Conditional statement (*1) if it contains ON OVERFLOW phrase, ON EXCEPTION phrase, or NOT ON EXCEPTION phrase.

Statement	Difference Between Conditional and Imperative Statements
	Imperative statement otherwise.
CANCEL statement	Imperative statement
CLOSE statement	Imperative statement
COMPUTE statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
CONTINUE statement	Imperative statement
DELETE statement	Conditional statement (*1) if it contains INVALID KEY phrase or NOT INVALID KEY phrase. Imperative statement otherwise.
DISPLAY statement	Conditional statement (*1) if it contains ON EXCEPTION phrase or NOT ON EXCEPTION phrase.
	Imperative statement otherwise.
DIVIDE statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
ENTRY statement	Imperative statement otherwise.
EVALUATE statement	Conditional statement (*1)
EXIT statement	Imperative statement otherwise.
EXIT PERFORM statement	Imperative statement otherwise.
EXIT PROGRAM statement	Imperative statement otherwise.
GENERATE statement	Imperative statement otherwise.
GO TO statement	Imperative statement otherwise.
IF statement	Conditional statement (*1)
INITIALIZE statement	Imperative statement otherwise.
INITIATE statement	Imperative statement otherwise.
INSPECT statement	Imperative statement otherwise.
MERGE statement	Imperative statement otherwise.
MOVE statement	Imperative statement otherwise.
MULTIPLY statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
OPEN statement	Imperative statement otherwise.
PERFORM statement	Imperative statement otherwise.
READ statement	Conditional statement (*1) if it contains AT END phrase, NOT AT END phrase, INVALID KEY phrase, or NOT INVALID KEY phrase. Imperative statement otherwise.
RELEASE statement	Imperative statement otherwise.
RETURN statement	Conditional statement (*1)

Statement	Difference Between Conditional and Imperative Statements
REWRITE statement	Conditional statement (*1) if it contains INVALID KEY phrase or NOT INVALID KEY phrase. Imperative statement otherwise.
SEARCH statement	Conditional statement (*1)
SET statement	Imperative statement otherwise.
SORT statement	Imperative statement otherwise.
START statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
STOP statement	Imperative statement otherwise.
STRING statement	Conditional statement (*1) if it contains ON OVERFLOW phrase or NOT ON OVERFLOW phrase. Imperative statement otherwise.
SUBTRACT statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
SUPPRESS statement	Imperative statement otherwise.
TERMINATE statement	Imperative statement otherwise.
UNLOCK statement	Imperative statement otherwise.
UNSTRING statement	Conditional statement (*1) if it contains ON OVERFLOW phrase or NOT ON OVERFLOW phrase. Imperative statement otherwise.
WRITE statement	Conditional statement (*1) if it contains INVALID KEY phrase, NOT INVALID KEY phrase, END-OF-PAGE phrase, or NOT END-OF-PAGE phrase. Imperative statement otherwise.

*1: This statement is changed to an imperative statement by putting an explicit scope terminator at the end of the statement.

A statement that explicitly transfers control to a following non executable statement (e.g., a paragraph name) is called a procedure branching statement. Procedure branching statements include the following:

- ALTER statement
- CALL statement
- EXIT statement
- EXIT PERFORM statement
- EXIT PROGRAM statement
- GO TO statement
- MERGE statement containing the OUTPUT PROCEDURE phrase
- PERFORM statement
- SORT statement containing the INPUT PROCEDURE phrase or OUTPUT PROCEDURE phrase

Sentence

The three types of sentence are conditional, imperative, and compiler-directing sentences. Each sentence type is constructed with different types of statements.

A conditional sentence is a conditional statement that ends with a period (separator). One conditional statement can be combined with several imperative statements to form one conditional sentence.

An imperative sentence is an imperative statement that ends with a period (separator). Several imperative statements can be combined to form one imperative sentence.

A compiler-directing sentence is a compiler-directing statement that ends with a period (separator). A compiler-directing sentence must consist of only one compiler-directing statement and a period (separator).

Scope of Statements

Use an explicit scope terminator or a period (separator) to explicitly specify the scope of a statement. Explicit scope terminators include the following words:

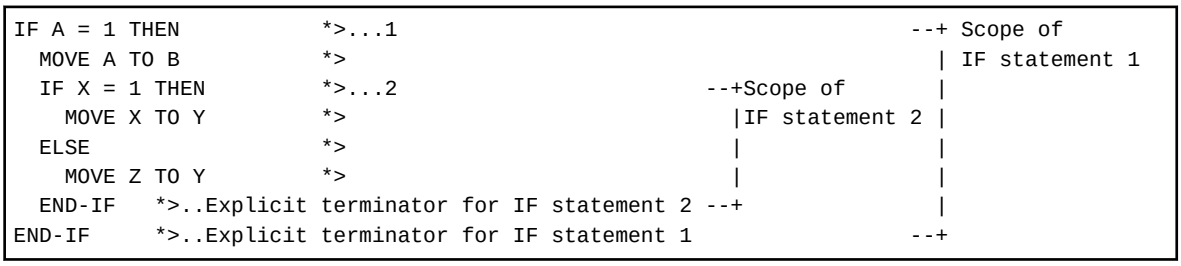
- END-ADD, END-CALL, END-COMPUTE, END-DELETE, END-DIVIDE, END-EVALUATE, END-IF, END-MULTIPLY, END-PERFORM, END-READ, END-RETURN, END-REWRITE, END-SEARCH, END-START, END-STRING, END-SUBTRACT, END-UNSTRING, END-WRITE

F

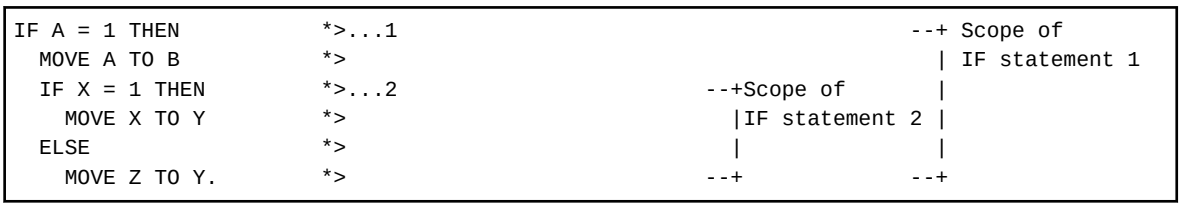
- END-ACCEPT, END-DISPLAY

An explicit scope terminator terminates the scope of the statements corresponding to the terminator.

An example of specifying the scope of statements with explicit scope terminators is shown below.



A period (separator) terminates the scope of all statements before the period. An example of specifying the scope of statements with a period (separator) is shown below.



6.2 Procedure Division Header

The procedure division must begin with a procedure division header. If the USING phrase is specified, a parameter can be received from the calling program.

F

If the RETURNING phrase is specified, a result can be returned to the called program.

Format

PROCEDURE DIVISION

[WITH { C | PASCAL | STDCALL } LINKAGE]

[USING {data-name-1} ...]

[RETURNING data-name-2].

F

[Win16] [Win32] The WITH phrase can be specified.

The RETURNING phrase is available for [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF] and [Linux64].

For the procedure division header in [.NET], refer to "Procedure Division" in "Chapter 12 Microsoft .NET Support".

Syntax Rules

1. A procedure division header can contain a USING phrase only if the program is called by a CALL statement with a USING phrase.

F However, in this implementation of COBOL, a procedure division header can also contain a USING phrase if the program executed first in the run unit receives execution time parameters.

2. In the USING phrase, specify the data-names corresponding to the parameters passed by the calling program. The items in USING phrase of a CALL statement in the calling program correspond to the data-names in USING phrase of the procedure division header of the called program. This is done in sequence from left to right, as they are specified in the USING phrase.
3. Data-name-1 must observe the following rules:
    a. Data-name-1 must be a data item defined in the LINKAGE SECTION.
    b. The level number of the data item for data-name-1 must be 01 or 77.
    c. The data description entry for data-name-1 must not contain the REDEFINES clause. However, data-name-1 can be the object of a REDEFINES clause.
4. Each data-name in a list must be unique.
5. For USING phrase restrictions, see "Inter-program Communication" in Appendix B: System Quantitative Restrictions.

F 6. The RETURNING phrase can be used in a program definition.

7. Data-name-2 must be defined as a data item with a level number 01 or 77 in the LINKAGE SECTION. A data description entry for data-name-2 must not specify the REDEFINES clause, however data-name-2 may be the object of a REDEFINES clause.
8. If the Procedure Division has an ENTRY statement, the Procedure Division header cannot specify the RETURNING phrase.
9. When RETURNING phrase is described in Procedure division header, group item with same kind of floating-point data items cannot be specified for data-name-2 in [LinuxIPF].

F

Rules for the WITH Phrase

1. The WITH phrase specifies the calling conventions for a called program. If the WITH phrase is omitted, the COBOL calling conventions are applied.
2. The WITH phrase must not be written in the Procedure Division header of a nested program.
3. The WITH phrase can only be used in the Procedure Division header of a program definition.

General Rules

1. The parameters in a CALL statement that specifies the BY CONTENT phrase, once the CALL statement is executed, can be referenced by data-name-1 in the called program. However, if the called program updates the value in data-name-1, the calling program cannot receive the updated value. The following must be the same in the use and character position count:
    - A data item in USING of the procedure division header
    - The corresponding data item in a BY CONTENT phrase of a CALL statement

2. The parameters in a CALL statement that specifies the BY REFERENCE phrase, once the CALL statement is executed, can be referenced by data-name-1 in the called program. If the called program updates the value in data-name-1, this updated value can be received by the calling program.

   The following must be the same in use and length:

   - A data item in USING of the procedure division header
   - The corresponding data item in BY REFERENCE of a CALL statement

F 3. The parameters in a BY VALUE phrase of a CALL statement cannot be received by the USING phrase of the procedure division header.

4. The contents of data-name-1 are always referenced based on the data description of data-name-1.

5. A data item defined in the LINKAGE SECTION of the called program can be used in the Procedure Division under any of the following conditions:

   a. The data item is specified in the USING phrase of

      - the Procedure Division header
F   - or an ENTRY statement.

   b. The data item is subordinate to the data item satisfying the condition in (a).

   c. A data item satisfying the condition in (a) or (b) is the object of a REDEFINES or RENAMES clause.

   d. The data item is subordinate to the data item satisfying the condition in (c).

   e. The data item is a condition-name or an index-name for a data item satisfying one of the conditions in (a) to (d).

F 6. If the RETURNING phrase is specified, a value will not be returned to the special register PROGRAM-STATUS of the calling program.

F **Rules for the WITH Phrase**

1. The calling conventions between the calling and called programs must be the same. If they are different, the execution result is undefined.
2. In a single separately compiled program, the Procedure Division header of the outermost program must be the same as the WITH phrase in the ENTRY statement.

6.3 Common Statement Rules

This section explains common statement rules and common specifications of statements.

6.3.1 Arithmetic Expressions

An arithmetic expression is formed by combining arithmetic operators and identifiers or literals. There are five types:

1. An arithmetic expression consisting of

   - a numeric data item,
F - a floating-point data item,
   - a numeric function,
   - an integer function,
   - a numeric literal,
   - or the figurative constant ZERO

2. An arithmetic expression formed by combining elements in (1) using arithmetic operators
3. An arithmetic expression formed by combining two arithmetic expressions using an arithmetic operator

4. Parenthesized arithmetic expression
5. An arithmetic expression preceded by a unary arithmetic operator

Arithmetic Operator

Arithmetic operators include binary and unary arithmetic operators. The following table lists the arithmetic operators:

Classification	Operator	Meaning
Binary arithmetic	+	Addition
operator	-	Subtraction
	*	Multiplication
	/	Division
	**	Exponentiation (*1)
Unary arithmetic	+	Same as multiplication by +1 (numeric literal).
operator	-	Same as multiplication by -1 (numeric literal).

*1: If positive and negative real numbers are produced by evaluating an exponentiation expression, the positive number overrides the negative number.

Rules for Writing an Arithmetic Expression

1. The following table lists the combinations of identifiers, literals, arithmetic operators, and parentheses permitted in an arithmetic expression:

		Subsequent Element				
		Identifier or Literal	(	Unary Arithmetic Operator +,-	Binary Arithmetic Operator +,-,*,/,**	)
Preceding Element	**Unary arithmetic operator +,-**	o	o	-	-	-
	(	o	o	o	-	-
	Binary arithmetic operator +, -, *, /, **	o	o	o	-	-
	)	-	-	-	o	o
	Identifier or literal	-	-	-	o	o

o: The subsequent element can be written.

-: The subsequent element cannot be written.

2. An arithmetic expression must begin with a unary arithmetic operator, a left parenthesis, an identifier, or a literal. It must end with a right parenthesis, an identifier, or a literal.
3. An identifier in an arithmetic expression must be
   - a numeric data item,
   - a floating-point data item,
   - a numeric function,

- or an integer function.

A literal in an arithmetic expression must be a numeric literal or the figurative constant ZERO.

4. An arithmetic operator must be preceded and followed by a space. However, a space between an arithmetic operator and a parenthesis can be omitted.
5. One left parenthesis must correspond to one right parenthesis. A left parenthesis must precede the corresponding right parenthesis. A unary arithmetic operator may appear first in an arithmetic expression following an identifier or another operator. If so, the unary arithmetic operator must be preceded by a left parenthesis.

Rules for Evaluating an Arithmetic Expression

1. The order of evaluating an arithmetic expression can be specified by using parentheses. Without parentheses, arithmetic operations are performed in the following order of precedence:
   - First: + and - (unary arithmetic operator)
   - Second: **
   - Third: * and /
   - Fourth: + and - (binary arithmetic operator)
2. Use parentheses to:
   a. Change the order of arithmetic operations on the same level.
   b. Change the order of arithmetic operations on different levels.
3. A parenthesized arithmetic expression is evaluated first. If there are parentheses within parentheses, the innermost parenthesized arithmetic expression is evaluated first.
4. Consecutive arithmetic operations on the same level are performed in order from left to right.

F

6.3.2 Boolean Expressions

A Boolean expression is formed by combining Boolean operators and identifiers or literals. It has one of the following five types:

1. A Boolean expression consisting of a Boolean data item, a Boolean literal, the figurative constant ZERO, or ALL Boolean literal.
2. A Boolean expression formed by combining elements in (1) using Boolean operators
3. A Boolean expression formed by combining two Boolean expressions using a Boolean operator
4. A Parenthesized Boolean expression
5. A Boolean expression preceded by the unary Boolean operator NOT

For details about how to use the examples, refer to "Boolean Expression" in the "Syntax Samples".

Boolean Operators

Boolean operators include binary and unary Boolean operators. The following table lists Boolean operators:

Classification	Operator	Meaning
Binary Boolean operator	AND	Boolean product
	OR	Boolean sum
	EXOR	Exclusive Boolean sum
Unary Boolean operator	NOT	Boolean negation

Rules for Writing a Boolean Expression

1. The following table lists combinations of identifiers, literals, Boolean operators, and parentheses permitted in a Boolean expression:

		Subsequent Element				
		Identifier or Literal	(	Unary Boolean Operator NOT	Binary Boolean Operator AND,OR, EXOR	)
Preceding Element	**Unary Boolean operator NOT**	o	o	-	-	-
	(	o	o	o	-	-
	Binary Boolean operator AND,OR,EXOR	o	o	o	-	-
	)	-	-	-	o	o
	Identifier or literal	-	-	-	o	o

   o: The subsequent element can be written.

   -: The subsequent element cannot be written.

2. A Boolean expression must begin with a unary Boolean operator, a left parenthesis, an identifier, or a literal. It must end with a right parenthesis, an identifier, or a literal.
3. An identifier in a Boolean expression must be a Boolean data item. A literal in a Boolean expression must be a Boolean literal, the figurative constant ZERO, or ALL Boolean literal.
4. All Boolean data items or literals in a Boolean expression must have the same length.
5. A Boolean operator must be preceded and followed by a space. However, a space between a Boolean operator and a parenthesis can be omitted.
6. One left parenthesis must correspond to one right parenthesis. A left parenthesis must precede a right parenthesis.

Rules for Evaluating a Boolean Expression

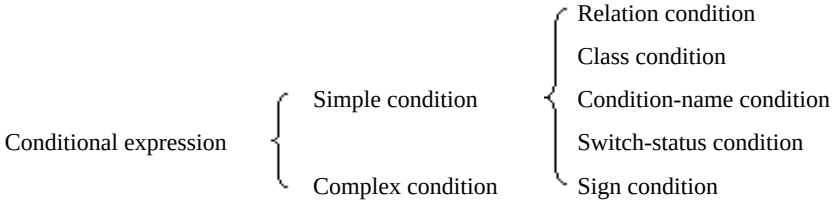
1. The order of evaluating a Boolean expression can be specified by using parentheses. Without parentheses, Boolean operations are performed in the following order of precedence:
   - First: NOT
   - Second: AND
   - Third: OR and EXOR
2. Use parentheses to:
   a. Change the order of Boolean operations on the same level.
   b. Change the order of Boolean operations on different levels.
3. A parenthesized Boolean expression is evaluated first. If there are parentheses within parentheses, the innermost parenthesized Boolean expression is evaluated first.
4. Consecutive Boolean operations on the same level are performed in order from left to right.

5. AND, OR, EXOR, and NOT in the following are Boolean operators, and not logical operators:
   - Two Boolean expressions combined only by AND, OR, or EXOR
   - One Boolean expression beginning with NOT

6.3.3 Conditional Expressions

Conditional expressions identify conditions that are tested to enable the object program to select between alternate paths of control depending on the truth value of the condition. A conditional expression can be specified in the EVALUATE, IF, PERFORM, and SEARCH statements.

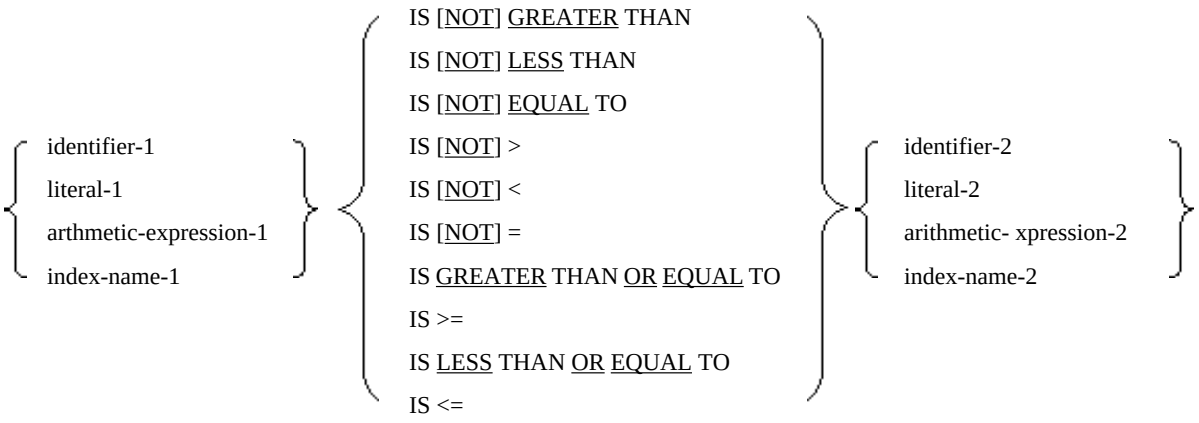
Conditional expressions have the following types:



6.3.3.1 Relation Condition

A relation condition compares two operands. The operands specified in a relation condition can be an identifier, a literal, an arithmetic expression, or an index-name.

Format



Note

>, <, =, >=, and <= are not underlined for avoiding confusion with other symbols.

= TO, > THAN and < THAN (combinations of symbols and words) are also accepted.

1. An operand on the left of a relational operator is called "left side of a condition." An operand on the right of a relational operator is called "right side of a condition".
2. A relation condition must contain one or more variables.
3. A reserved word making up a relational operator must be preceded and followed by one or more spaces.
4. The truth value is determined by comparing the values on the right and left sides of a relation condition. This is done according to the relational operator. The relational operator determines the type of relation condition comparison. The following table lists the meanings of relational operators:

Relational Operator	Meaning
IS [NOT] GREATER THAN and IS [NOT] >	The value on the left side (subject) of an expression is greater than that on the right side (object). [The value on the left side (subject) of an expression is not greater than that on the right side (object).] (*1)
IS [NOT] LESS THAN and IS [NOT] <	The value on the left side of an expression is less than that on the right side. [The value on the left side (subject) of an expression is not less than that on the right side (object).] (*1)
IS [NOT] EQUAL TO and IS [NOT]=	The value on the left side of an expression equals that on the right side. [The value on the left side (subject) of an expression is not equal to that on the right side (object).] (*1)
IS GREATER THAN OR EQUAL TO and IS >=	The value on the left side of an expression is greater than or equal to that on the right side object. (*2)
IS LESS THAN OR EQUAL TO and IS <=	The value on the left side subject of an expression is less than or equal to that on the right side object. (*3)

*1: Brackets [ ] indicate that NOT is written within the brackets.

*2: IS GREATER THAN OR EQUAL TO is equivalent to IS NOT LESS THAN. IS >= is equivalent to IS NOT <.

*3: IS LESS THAN OR EQUAL TO is equivalent to IS NOT GREATER THAN. IS <= is equivalent to IS NOT >.

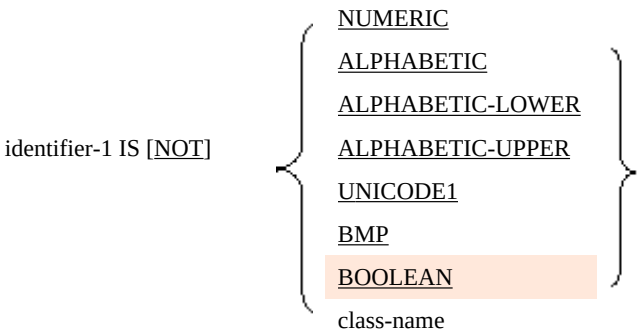
5. For details of the rules for relation condition comparison, see the topic titled "Comparison rules" later in this section.

6.3.3.2 Class Condition

A class condition is for examining the class of a data item.

For details about how to use the examples, refer to "Class Condition" in the "Syntax Samples".

Format



For class conditions in [.NET], refer to "Class Condition" in "Chapter 12 Microsoft .NET Support".

1. In a class condition, the content of identifier-1 is examined according to a class examination specification and a truth value is determined. The following table lists the conditions under which class examinations show that an expression is true:

Type of Class Examination	Condition Where a Class Examination Shows that an Expression Is True
NUMERIC examination	identifier-1 contains only digits 0 to 9, or digits 0 to 9 with operational signs. (*1)
ALPHABETIC examination	identifier-1 contains upper case alphabetic characters A to Z, and blanks, lower case alphabetic characters a to z, and blanks, or upper case and lower case alphabetic characters, and blanks.
ALPHABETIC-LOWER examination	identifier-1 contains lower case alphabetic characters a to z and blanks.
ALPHABETIC-UPPER examination	identifier-1 contains upper case alphabetic characters A to Z, and blanks.
UNICODE1 examination	identifier-1 contains characters provided for by UNICODE1.1(ISO/IEC 10646-1:1993).
BMP examination	identifier-1 contains characters provided for by ISO/IEC 10646-1.
BOOLEAN examination	identifier-1 contains only Boolean characters 0 and 1.
class-name examination	identifier-1 contains a set of the characters specified in the CLASS phrase of a special-names paragraph

*1: For details, see item (6).

2. Identifier-1 must be one of the following:
   - Data item with USAGE DISPLAY
   - Alphanumeric function identifier
   - Packed decimal data item
3. For the NUMERIC examination, the following data items cannot be specified in identifier-1:
   - Alphabetic data item
   - Group item containing signed numeric data items
   - National data item
   - National edited data item
   - Boolean data item
4. For the ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, and class-name examinations, the following data items cannot be specified in identifier-1:
   - Numeric data item
   - National data item
   - National edited data item
   - Boolean data item
5. For the BOOLEAN examination, the following data items cannot be specified in identifier-1:
   - Numeric data item
   - Alphabetic data item
   - National data item
   - National edited data item
6. The NUMERIC examination shows that an expression is true under any of the following conditions:
   a. Identifier-1 contains only numeric digits without operational signs. Alternatively, identifier-1 (packed decimal data item) contains hexadecimal F as a sign.

b. Identifier-1 contains only numeric digits with a valid operational sign. An operational sign is valid under one of the following conditions:

   - A SEPARATE CHARACTER phrase is specified in the SIGN clause of identifier-1, and the sign is "+" or "-" expressed in the standard data format.
   - A SEPARATE CHARACTER phrase is not specified in the SIGN clause of identifier-1 (zoned decimal data item), and the sign is hexadecimal 4, 5, or 3.
   - Identifier-1 (packed decimal data item) contains hexadecimal C, D, or F as a sign.

7. NOT and a key word are combined to form one class condition. An expression with NOT has a truth value opposite from that of the expression without NOT. For example, NOT NUMERIC is true if the operand is not a number.
8. For the UNICODE1 and BMP examination, the following data items cannot be specified in identifier-1:

   - Numeric data item
   - Boolean data item

F

9. The UNICODE1 and BMP examination can be used when the encoding form of identifier-1 is not SJIS.

6.3.3.3 Condition-name

A condition-name is for checking whether the value of a conditional variable equals one of the values associated with a condition-name.

Format

condition-name-1

1. A condition-name is true under either of the following conditions:
   a. THRU is not specified in the VALUE clause of condition-name-1 (data description entry). Also, the value of a conditional variable equals that specified in the VALUE clause.
   b. THRU is written in the VALUE clause of condition-name-1 (data description entry). Also, the value of a conditional variable is within the range (including the lower and upper limits specified in THRU) specified in the VALUE clause.
2. Compare the value of a conditional variable with that of condition-name-1 according to the rules for a relation condition comparison. For details of the comparison rules, see the topic titled "Comparison rules" later in this section.

Example

Examples of condition-names are shown below.

a. If conditional variables and condition-names are defined as follows:

```
02  MONTH  PICTURE  99.          *>... Definition of the conditional variable MONTH
  88  SPRING-M  VALUE  3  THRU   5.  *>     --+ Definitions of the condition-names
  88  SUMMER-M  VALUE  6  THRU   8.  *>       | SPRING-M, SUMMER-M, FALL-M, and
  88  FALL-M    VALUE  9  THRU  11.  *>       | WINTER-M
  88  WINTER-M  VALUE  12, 1, 2.     *>     --+
```

b. In order to check the value of the conditional variable MONTH, you can use condition-names to write the following IF statements:

   1. IF SPRING-M: Equivalent to IF MONTH >= 3 AND <= 5.
   2. IF SUMMER-M: Equivalent to IF MONTH >= 6 AND <= 8.
   3. IF FALL-M: Equivalent to IF MONTH >= 9 AND <= 11.
   4. IF WINTER-M: Equivalent to IF MONTH = 12 OR 1 OR 2.

6.3.3.4 Switch-status Condition

A switch-status condition is for checking the status of an external switch.

Format

condition-name-1

1. Define an external switch in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. Associate the condition-name to the on or off status of the switch.
2. If the external switch is set to condition-name-1, the switch-status condition is true.

6.3.3.5 Sign Condition

A sign condition is for checking whether the algebraic value of an arithmetic expression is greater than, less than, or equal to zero.

Format

arithmetic-expression-1 IS [NOT] { POSITIVE / NEGATIVE / ZERO }

1. Arithmetic-expression-1 must contain at least one identifier.
2. In a sign condition, the algebraic value of an arithmetic expression is examined, and the truth value is determined. This is done based on whether POSITIVE, NEGATIVE, or ZERO is specified. The following table lists the meanings of these key words:

Keyword	Meaning
POSITIVE	Greater than zero
NEGATIVE	Less than zero
ZERO	Equal to zero

3. NOT and a key word are combined to form one sign condition. An expression with NOT has the truth value opposite to that of the expression without NOT. For example, NOT ZERO is true if the value of an arithmetic expression is not zero (i.e., positive or negative).

6.3.3.6 Complex Condition

A complex condition is formed by combining one or more simple conditions and logical operator.

A complex condition consists of negated and combined conditions. A negated condition is a simple or complex condition preceded by the logical operator NOT. A combined condition is simple or complex conditions combined by the logical operator AND or OR.

The truth value of a complex condition is obtained by evaluating individual conditions and performing all logical operations sequentially.

The following table lists the meanings of logical operators:

Logical Operator	Meaning	Truth Value
NOT	Logical negation	If the condition following NOT is false, the truth value is true. If the condition following NOT is true, the truth value is false.
AND	Logical product	If both conditions on the right and left of AND are true, the truth value is true.

Logical Operator	Meaning	Truth Value
		If either or both conditions on the right and left of AND are false, the truth value is false.
OR	Logical sum	If either or both conditions on the right and left of OR are true, the truth value is true. If both conditions on the right and left of OR are false, the truth value is false.

Negated Condition

A negated condition is for reversing the truth value of a condition.

Format

NOT conditon-1

Combined Condition

A combined condition is for obtaining a logical product or sum.

Format

condition-1 { { AND / OR } condition-2 } ...

Rules for Writing a Complex Condition Expression

1. The following table lists combinations of simple conditions, logical operators, and parentheses permitted in a complex condition expression:

		Subsequent Element					
		Simple Condition	AND	OR	NOT	(	)
Preceding Element	**Simple Condition**	-	o	o	-	-	o
	AND	o	-	-	o	o	-
	OR	o	-	-	o	o	-
	NOT	o	-	-	-	o	-
	(	o	-	-	o	o	-
	)	-	o	o	-	-	o

   o : The subsequent element can be written.

   - : The subsequent element cannot be written.

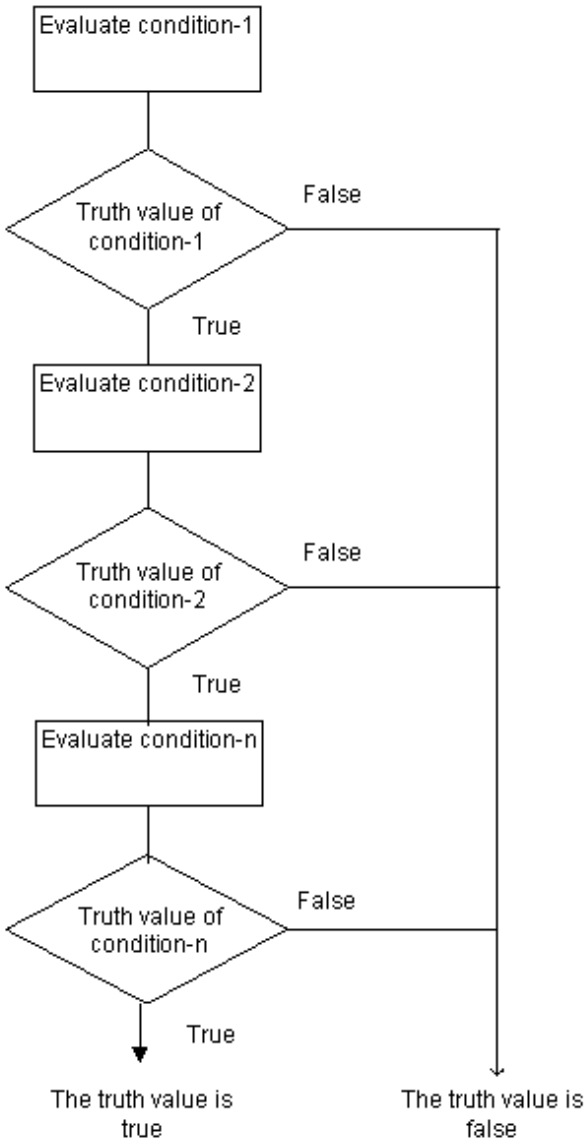
2. A complex condition must begin with a simple condition, NOT, or left parenthesis and end with a simple condition or right parenthesis.
3. A logical operator must be preceded and followed by a space. However, a space between a logical operator and a parenthesis can be omitted.
4. One left parenthesis must correspond to one right parenthesis. A left parenthesis must precede the corresponding right parenthesis.

Rules for Evaluating a Complex Condition

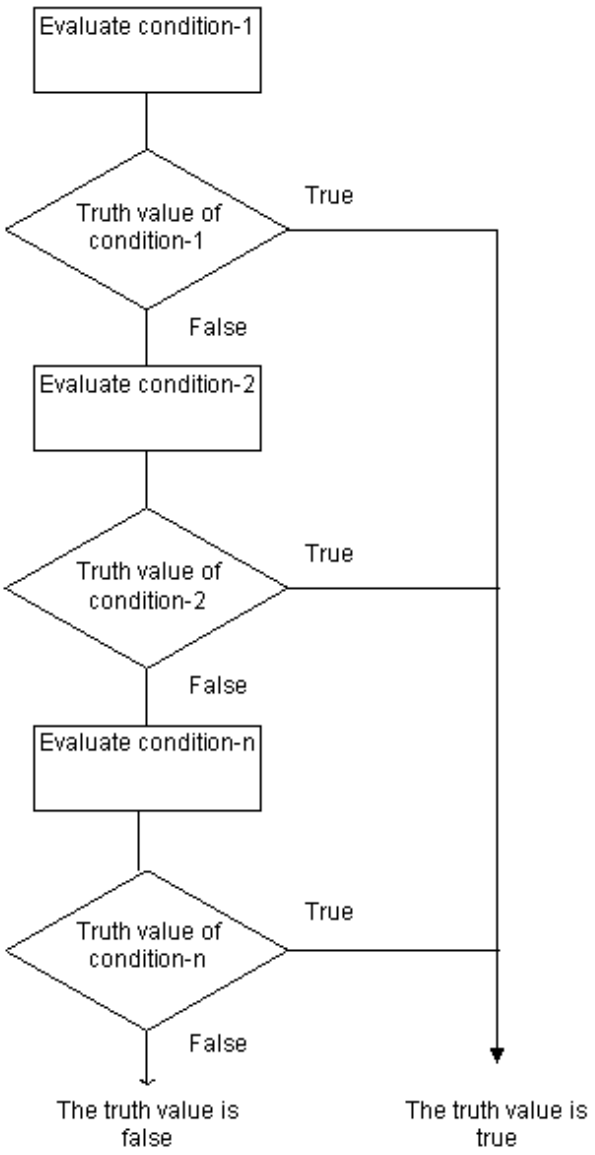
1. The order of evaluating a complex condition can be specified by using parentheses. Without parentheses, logical operations are performed in the following order of precedence:
   - First: NOT
   - Second: AND
   - Third: OR
2. To change the order of performing logical operations, enclose the range of the conditions combined by a logical operator in parentheses. The parenthesized conditions combined by a logical operator are evaluated first, or a parenthesized logical operation is performed first. If there are parentheses within parentheses, the innermost parenthesized condition is evaluated first, or the innermost parenthesized logical operation is performed first.
3. Consecutive logical operations on the same level are performed in order from left to right.
4. Evaluation of the individual conditions of a complex condition terminates when the truth value of the complex condition is determined. This is done regardless of whether all the individual conditions are evaluated.
5. The values of the arithmetic expressions and functions in a complex condition expression are determined when the expression is evaluated.
6. A negated condition is evaluated when the condition combined with the logical operator NOT is evaluated.

Example of the Order of Evaluating a Complex Condition

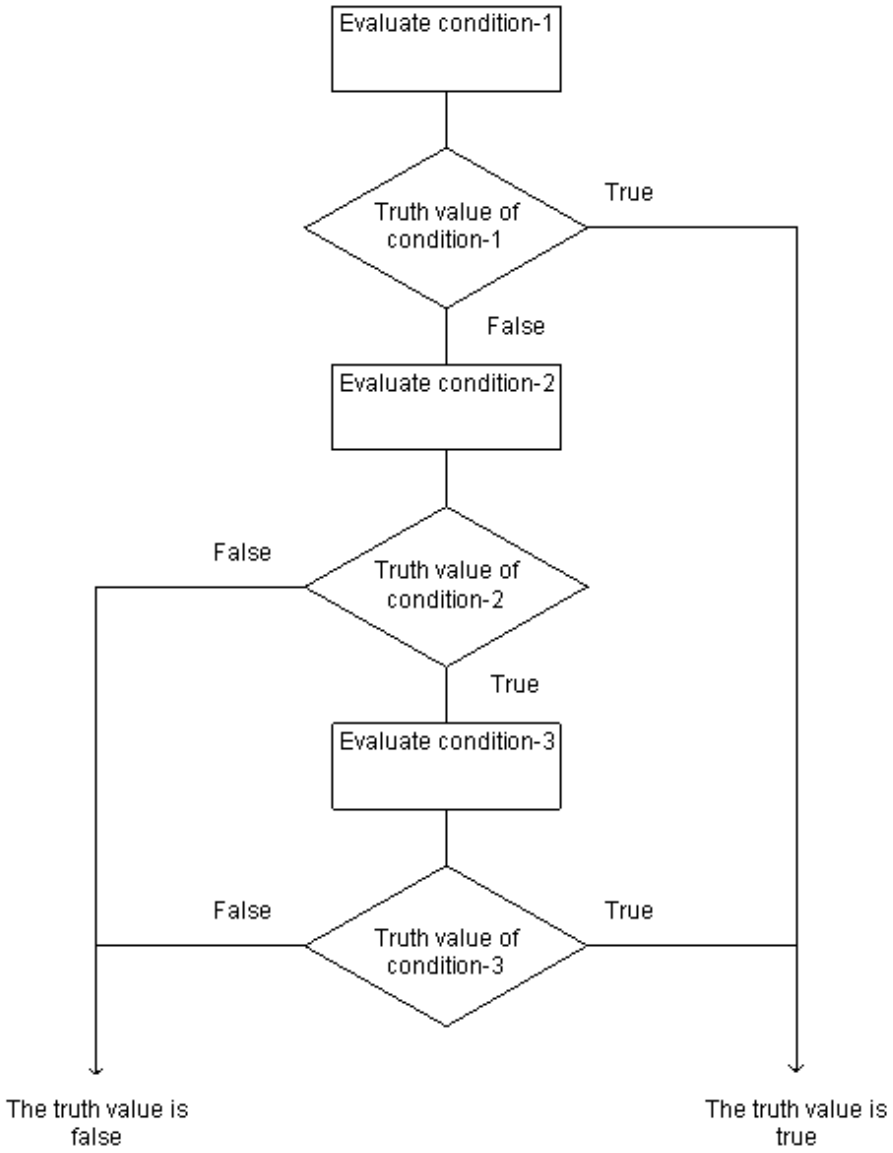
The following figure shows the order of evaluating "condition-1 AND condition-2 AND ... condition-n":



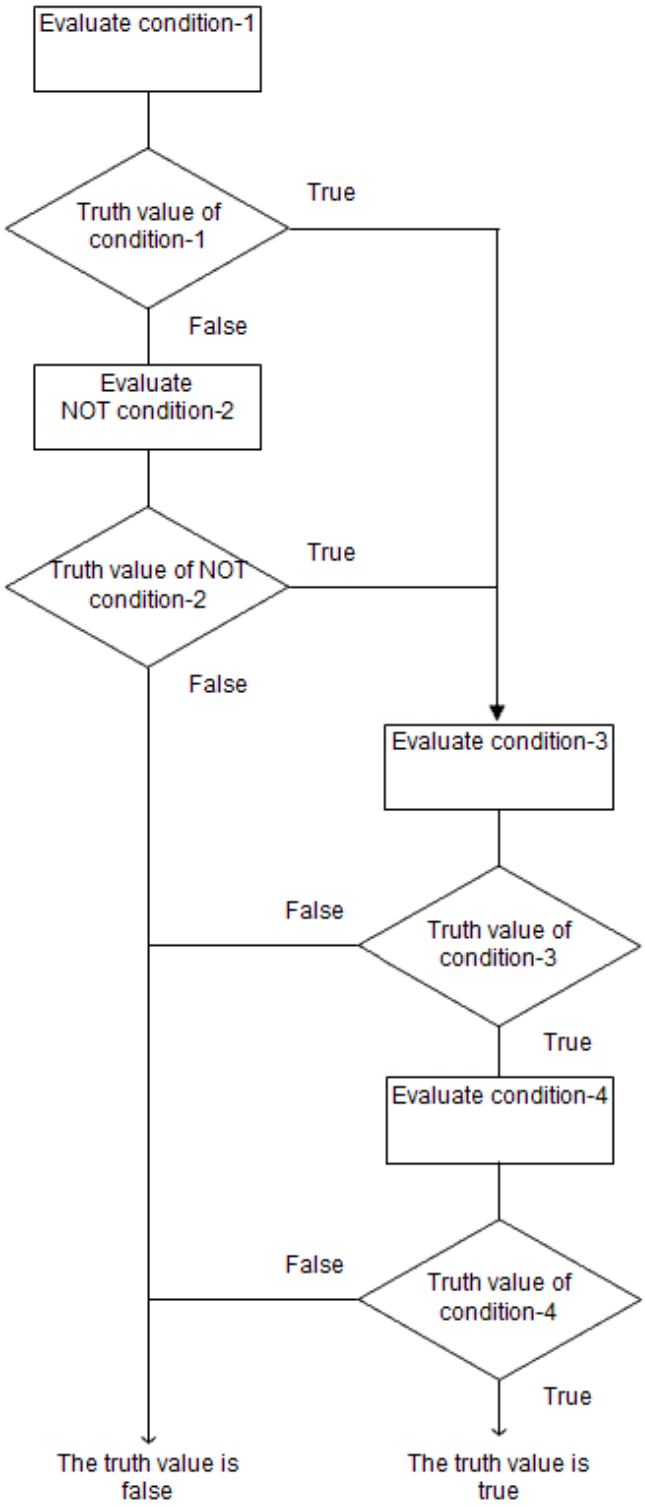
The following figure shows the order of evaluating "condition-1 OR condition-2 OR ... condition-n":



The following figure shows the order of evaluating "condition-1 OR condition-2 AND condition-3":



The following figure shows the order of evaluating "(condition-1 OR NOT condition-2) AND condition-3 AND condition-4":



6.3.3.7 Abbreviating a Combined Relation Condition

A complex condition formed by combining a relation condition and a logical operator is called a combined relation condition. A combined relation condition expression may not contain a parenthesis that changes the order of performing logical operations. If so, part of the combined condition can be omitted.

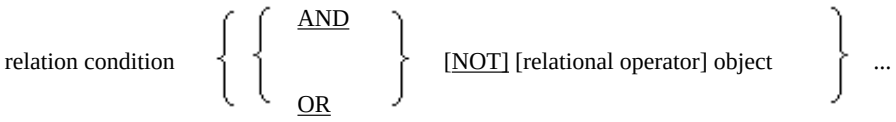
In a statement, part of a combined condition can be omitted as follows:

a. The subject of a second or subsequent relation condition can be omitted if it is the same as the subject of the preceding relation condition.

b. The subject and relational operator of a second or subsequent relation condition can be omitted if they are the same as the subject and relational operator.

For details about how to use the examples, refer to "Abbreviating Combined Relation Conditions" in the "Syntax Samples".

Format



1. These two methods of omission can be used for a series of relation conditions.
2. The subject of the preceding relational condition is used in place of the omitted subject. The preceding relational operator is used in place of the omitted relational operator. The omitted operands and relational operators are repeatedly compensated for until an unabbreviated simple condition appears in the complex condition expression. The results of compensating for the subject and relational operator must observe the rules provided in the section titled "Complex Condition" earlier in this section.
3. NOT in the expression for an abbreviated combined relation condition is interpreted as follows:
   a. NOT followed by GREATER, >, LESS, <, EQUAL, or = is treated as part of the relational operator.
   b. NOT other than that in (a) is treated as a logical operator. Therefore, the result of compensating for any omitted subject or omitted relational operator is a negated condition.
4. The following table lists examples of abbreviated combined relation conditions:

Abbreviated Combined Relation Condition	Unabbreviated Combined Relation Condition
a > b AND NOT < c OR d	((a > b) AND (a NOT < c)) OR (a NOT < d)
a NOT EQUAL b OR c	(a NOT EQUAL b) OR (a NOT EQUAL c)
NOT a = b OR c	(NOT (a = b)) OR (a = c)
NOT (a GREATER b OR < c)	NOT ((a GREATER b) OR (a < c))
NOT (a NOT > b AND c AND NOT d)	NOT (((a NOT > b) AND (a NOT > c)) AND (NOT (a NOT > d)))

_ : Operand on the left side and relational operator that can be omitted

6.3.4 Comparison Rules

This section explains the rules for relation condition comparison. One of the following rules is applied based on a combination of the subject and object of a relation condition:

- Nonnumeric comparison
- Numeric comparison
- Comparison of national characters
- Boolean comparison
- Comparison of pointer data
- Index comparison

The rules for Boolean comparison apply to the subject or object Boolean data item of a relation condition. The rules for pointer data comparison apply to the subject or object pointer data item or ADDR function of the condition.

The rules for index comparison apply to the subject or object index-name or index data item of the condition.

The following table lists combinations of the operands to which the rules for character, numeric, and national character comparison are applied:

For combinations of the operands in [.NET], refer to "Comparisons" in "Chapter 12 Microsoft .NET Support"

		Subject or object of condition						
		Group	Alphabetic character Alphanumeric(*1) Alphanumeric edited Numeric edited	National National Edited	Zoned Decimal	Binary Packed Decimal	Floating Point	Arithmetic expression(*2)
Object or subject of condition	Group Item	Nonnumeric comparison	Nonnumeric comparison	Nonnumeric comparison	Nonnumeric comparison	Nonnumeric comparison	Nonnumeric comparison	-
	Alphabetic data item Alphanumeric data item (*1) Alphanumeric edited data item Numeric edited data item Nonnumeric literal(*3)	Nonnumeric comparison	Nonnumeric comparison	-	Nonnumeric comparison	-	-	-
	National data item(*4) National edited data item National literal(*5)	Nonnumeric comparison	-	Comparison of national characters	-	-	-	-
	Figurative constant SPACE HIGH-VALUE LOW-VALUE	Nonnumeric comparison	Nonnumeric comparison	Comparison of national characters	Nonnumeric comparison	-	-	-
	Figurative constant QUOTE	Nonnumeric	Nonnumeric	-	Nonnumeric	-	-	-

	Symbolic-character	comparison	comparison		comparison			
	Zoned decimal item	Nonnumeric comparison	Nonnumeric comparison	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison
	Binary item packed decimal item	Nonnumeric comparison	-	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison
	Numeric literal	Nonnumeric comparison	Nonnumeric comparison (*6)	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison
	Figurative constant ZERO	Nonnumeric comparison	Nonnumeric comparison	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison
	Floating-point data item Floating-point literal	Nonnumeric comparison	-	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison
	Arithmetic expression(*2)	-	-	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison

- : Incomparable combination

*1 : Alphanumeric data items include alphanumeric functions.

*2 : Arithmetic expressions include numeric and integer functions.

*3 : Character literal include ALL character literals.

F

*4 : National data items include national functions.

*5 : National nonnumeric literal include ALL national nonnumeric literals.

*6 : An alphanumeric function cannot be compared with a numeric literal.

Nonnumeric Comparisons

The rules for nonnumeric comparison are as follows:

1. Two operands are compared based on the collating sequence.
2. The number of the character positions for two operands may be the same. If so, each pair of the characters in the corresponding positions is sequentially compared from left to right. The result of comparison is determined as follows:
    a. If all the corresponding characters are the same, the two operands are equal.
    b. Each pair of the characters in the corresponding positions is compared until a pair of different characters appears. At this point, comparison determines which operand including a high position character is the largest in the collating sequence.
3. The number of the character positions for two operands may differ. If so, the shorter operand is treated as if it were padded on the right with spaces to the length of the longer operand. The method of comparison is the same as for (2).

4. Suppose the following are compared:
    a. Numeric operand
        - zoned decimal data item
        - binary data item
        - packed decimal data item
        - numeric literal
        - figurative constant ZERO
        - floating-point data item
        - floating-point literal
    b. Nonnumeric operand
        - group item
        - alphabetic data item
        - alphanumeric data item
        - alphanumeric edited data item
        - numeric edited data item
        - nonnumeric literal
        - national data item
        - national edited data item
        - national literal
        - figurative constant SPACE, HIGH-VALUE, LOW-VALUE, or QUOTE
        - symbolic-character
    c. The following rules are applied:
        - The numeric operand must be an integer.
        - The nonnumeric operand may be an elementary item or a nonnumeric literal. If so, the numeric operand is treated as if it were copied to an alphanumeric data item having the same length as the nonnumeric operand. This alphanumeric data item is compared with the nonnumeric operand.
        - The nonnumeric operand may be a group item. If so, first the numeric operand is treated as if it were copied into a group item having the same length as the nonnumeric operand. Then, this group item is compared with the nonnumeric operand.
5. When either operand is a strongly-typed group item, the other operand must also be a group item that is strongly-typed with the same type.
6. When the two operands are alphanumeric data items or alphanumeric edited data items, then the encoding form of the two operands must match. This rule is specific to [Win32], [Winx64] and [Linux64].
7. For details of the collating sequence, see the topic "PROGRAM COLLATING SEQUENCE clause" in the "Configuration Section" section of Chapter 4, the "Environment Division".

Numeric Comparisons

The rules for numeric comparison are as follows:

1. Two operands are compared based on an algebraic value.
2. The value zero is compared as zero regardless of whether a sign is included or not.
3. An unsigned operand is treated as if the sign were positive for comparison.

F

Comparison of National Nonnumeric Characters

The rules for comparing national nonnumeric characters are as follows:

1. Two operands are compared based on the collating sequence of national nonnumeric characters.
2. The number of digits of the national nonnumeric characters for two operands may be the same. If so, each pair of the national nonnumeric characters in the corresponding positions is sequentially compared from left to right. The result of comparison is determined as follows:
    a. If all the corresponding national nonnumeric characters are the same, the two operands are equal.
    b. Each pair of the national nonnumeric characters in the corresponding positions is compared until a pair of different national nonnumeric characters appears. At this point, comparison determines which operand including a high-position national character is largest in the national character collating sequence.
3. The number of the national character positions for two operands may differ. If so, the shorter operand is treated as if it were padded on the right with national spaces to the length of the longer operand. The method of comparison is the same as for (2).
4. When one of the operands is a national item or national edited item, then the following rules are applied. This rule is unique to [Win32], [Winx64] and [Linux64].
    a. When the other operand is a National item or National edited item, then the encoding form of the two operands must be the same.
    b. When the other operand is a National non numeric literal or a symbolic-constant SPACE, then the encoding form of the literal becomes the encoding form of the operand of the National item or National edited item.
    c. When the other operand is a National hexadecimal non numeric literal, then the literal must be written in a big-endian value of the encoding form to be compared.

F

Boolean Comparisons

The rules for Boolean comparison are as follows:

1. For Boolean comparison, a Boolean data item can be compared with the following operands:
    - Boolean literal (including ALL Boolean literals)
    - Figurative constant ZERO
2. Boolean data items to be compared need not be for the same purpose.
3. A relational operator must be either of the following:
    - IS [NOT] EQUAL TO
    - IS [NOT] =
4. The length (number of Boolean characters) of two operands must be the same.
5. Each pair of the Boolean characters in the corresponding positions of two operands is sequentially compared from left to right. If all the corresponding Boolean characters are the same, the two operands are equal.
6. A Boolean expression cannot be written as an operand.

F

Comparison of Pointer Data

The rules for comparing pointer data are as follows:

1. For pointer data comparison, a pointer data item or an ADDR function can be compared with the following operands:
    - Pointer data item
    - ADDR function
    - Figurative constant ZERO

2. Pointer data comparison can be performed only in the relation condition expression of the IF or EVALUATE statement.
3. A relational operator must be either of the following:
   - IS [NOT] EQUAL TO
   - IS [NOT] =

Index Comparison

The following table lists the rules for index comparison:

		Subject or Object of a Condition			
		Index-name	Index Data Item	Numeric Literal (Integer Only)	Numeric Data Item (Integer Only)
Object or Subject of a Condition	**Index-name**	Comparison of occurrence numbers	Comparison without conversion	Comparison between an occurrence number and an integer	Comparison between an occurrence number and an integer
	Index data item	Comparison without conversion	Comparison without conversion	-	-

-: Incomparable combination

Comparison of the occurrence numbers: The occurrence numbers corresponding to the index-names are compared.

Comparison between the occurrence number and the integer: The occurrence number corresponding to the index-name is compared with another operand.

Comparison without conversion: Actual values are compared as they are.

Note that an arithmetic expression can be used in the comparison.

6.3.5 Rules for Moving (Transcribing) Data

This section explains the rules for moving data.

Data is not truly moved, but rather, transcribed, as the source field is not changed.

Execution of a statement may cause movement of a data item, a literal, or the result of an arithmetic operation into another data item. The rules for the movement of data apply not only to such implicit MOVE statements, but also to explicit MOVE statements.

There are two categories of transcription rules, one for elementary items and one for group items. If either or both the sending and receiving sides are group items, the rules for moving group items are applied. Otherwise, the rules for moving elementary items are applied.

Moving Elementary Items

When moving elementary items, one of the following rules is applied depending on the category and use of the receiving item:

- Alphabetic copying
- Alphanumeric and alphanumeric edited copying
- Numeric and numeric edited copying

F

- Floating-point copying
- National and national edited copying
- Boolean copying

- Pointer data copying

When moving an elementary item, the internal representation format is converted, edited, or de-editing, if necessary.

The following table lists the combinations of operands for moving elementary items:

			Receiving Side						
			Alphabetic character	Alphanumeric and Alpha-Numeric Edited	Numeric and Numeric Edited	Floating Point	National and National Edited	Boolean	Pointer Data
Sending Side	Alphabetic item		[1]	[2]	-	-	-	-	-
	Alphanumeric item (*1) Nonnumeric literal (*2)		[1]	[2]	[3]	[4]	-	[6] (*8)	-
	Alphanumeric edited data item		[1]	[2]	-	-	-	-	-
	Numeric item and	Integer	-	[2] (*9)	[3]	[4]	-	-	-
	literal	Non-integer	-	-	[3]	[4]	-	-	-
	Numeric edited data item		-	[2]	[3[	[4]	-	-	-
	Floating-point item Floating-point literal		-	-	[3]	[4]	-	-	-
	National item (*3) National edited item National literal (*4)		-	-	-	-	[5]	-	-
	Boolean item Boolean literal (*5)		-	[2] (*7)	-	-	-	[6]	-
	Pointer data item (*6)		-	-	-	-	-	-	[7]
	Figurative constant ZERO		-	[2]	[3]	[4]	-	[6]	[7]
	Figurative constant SPACE		[1]	[2]	-	-	[5]	-	-
	Figurative constants HIGH-VALUE LOW-VALUE		-	[2]	-	-	[5]	-	-
	Figurative constant QUOTE Symbolic-character		-	[2]	-	-	-	-	-

- [1] to [7] : Combination of items that can be moved, and are explained in the headings below
- - : Combination of items that cannot be moved

*1 Alphanumeric data items include alphanumeric functions.

*2 Nonnumeric literals include ALL nonnumeric literals.

F

*3 National data items include national functions.

*4 National nonnumeric literals include ALL national nonnumeric literals.

*5 Boolean literal include ALL Boolean literals.

*6 Pointer data items include ADDR functions.

*7 A Boolean data item cannot be copied into an alphanumeric edited data item.

*8 A nonnumeric literal cannot be copied into a Boolean data item.

*9 There is an exception to the rule when the "Sending Side" is "Integer" and the "Receiving Side" is "Alphanumeric and Alphanumeric Edited". When the "Sending Side" is a BINARY-CHAR/SHORT/LONG/DOUBLE data item, this combination is not allowed.

[1] Alphabetic Transcription

If the receiving side is an alphabetic data item, data is aligned, and the remaining portion is padded with blanks as required. This is done according to the standard alignment rule.

[2] Alphanumeric and Alphanumeric Edited Transcription

If the receiving side is an alphanumeric data item or an alphanumeric edited data item, the following rules are applied. However, it is not applicable to format 3 of MOVE statement:

1. Data is aligned, and the remaining portion is padded with blanks as required. This is done according to the standard alignment rule.
2. If the sending side is a signed numeric data item, the sign is not transcribed. If the SIGN clause in the sending item contains a SEPARATE phrase, the sign is not transcribed. Therefore, in the edited data format, the number of the digits of the sending side is treated as if it were one digit less.
3. If the sending side is a numeric edited data item, data is not de-edited (removal of edited characters).
4. If use of the sending side differs from that of the receiving side, the representation of the sending side is converted to the internal representation of the receiving side.
5. If the PICTURE clause of the sending numeric data item contains P, the digit position indicated by P is treated as a zero. P is included in a calculation of the length of the sending side.
6. If the sending side is an alphanumeric data item or an alphanumeric edited data item, the encoding form of the sending side and receiving side must match. This rule is specific to [Win32], [Winx64] and [Linux64].

[3] Numeric and Numeric Edited Transcription

If the receiving side is a numeric data item or a numeric edited data item, the following rules are applied:

1. Data is aligned by the decimal point, and the remaining portion is padded with zeros as required. This is done according to the standard alignment rule. The zeros may be converted to other characters according to the description of the PICTURE clause.
   In this compiler, compile option TRUNC must be specified. When the binary item is set to the receiving side without specifying compile option TRUNC, the left end might not be truncated. The result of the move is not guaranteed when not installing on item on the receiving side because the left end is not truncated.
2. If the sending side is a numeric edited data item, the data is first de-edited, and an unsigned, unedited number is obtained. The unedited number is moved to the receiving side.
3. If the receiving side is a signed numeric data item, the receiving side has the same sign as the sending side. At this point, the representation format of the sign is converted, if necessary. If the sending side does not have a sign, the positive sign is assigned to the receiving side.
4. If the receiving side is an unsigned numeric data item, the absolute value of the sending side is transcribed. A sign is not assigned to the receiving side.
5. If the category of the sending side is alphanumeric, the send data is treated as an unsigned integer, and transcribed.
6. If the sending side is a floating-point data item and its value has a higher number of digits in the decimal part than the receiving side, the value rounded off to the number of digits in the decimal part of the receiving side is transcribed.

F

[4] Floating-point Transcription

If the receiving side is a floating-point data item, data is treated as if it were transcribed into a non-integer numeric data item.

F

[5] National and National Edited Transcription

If the receiving item is a National item or National edited item, then the following rules are applied. However, it is not applicable to format 3 of MOVE statement.

- In accordance with standard alignment rules, it is justified and blank space-filling is performed.
- When the sending item is a National item or National edited item, then the encoding of the sending item and receiving item must match. This rule is specific to [Win32], [Winx64] and [Linux64].
- When the sending item is a Japanese non numeric literal or figurative constant SPACE, then the value that matches the encoding of the receiving item is moved. This rule is specific to [Win32], [Winx64] and [Linux64].
- When the sending item is a National hexadecimal non numeric literal, then the literal value must be described in big-endian of the encoding of the receiving side. The value that matches the endian on the receiving item is moved. This rule is specific to [Win32], [Winx64] and [Linux64].

F

[6] Boolean Transcription

If the receiving side is a Boolean data item, the following rules are applied:

1. Data is aligned, and the remaining portion is padded with Boolean character zeros as required. This is done according to the standard alignment rule.
2. If the sending side is an alphanumeric data item, the sending side is treated as an external Boolean data item.

F

[7] Pointer Data Transcription

If the receiving side is a pointer data item, the following rules are applied:

1. If the sending side is a pointer data item or an ADDR function, the contents of the sending side are transcribed as they are.
2. If the sending side is the figurative constant ZERO, a zero is transcribed.

Moving Group Items

If either or both of the sending and receiving sides are group items, the following rules are applied:

F

1. A Boolean data item, a pointer data item, or a floating-point literal cannot be transcribed into a group item.
2. A group item cannot be transcribed into a Boolean data item or a pointer data item.
3. A group item is transcribed in the same way in which an elementary item is transcribed to and from an alphanumeric data item.
4. The internal representation format is not converted.
5. The usage of individual elementary items and group items belonging to the group are disregarded. The entire group item is transcribed as one alphanumeric data item. If an OCCURS clause is specified in a group item or in a data item belonging to a group item, another rule is applied. For details of the rules for copying a group item containing the OCCURS clause, see the topic titled "OCCURS clause" in the "Data Description Entry" section of Chapter 5, the "Data Division".

6.3.6 Arithmetic Statements

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements are called arithmetic statements. The following rules are common to arithmetic statements:

1. The data description entries of the operands in an arithmetic statement need not be the same. During computation, data is converted, and aligned by a decimal point as required.

2. An arithmetic operation may require the creation of a temporary operational result data item. This temporary data item is called an intermediate result. The intermediate result is created by the compiler as a signed numeric data item. The number of the digits of the intermediate result is based on the algorithm explained in Appendix D, "Intermediate Results". The operational result temporarily stored during execution is moved into the receiving data item according to the MOVE statement rules.

6.3.7 More Than One Arithmetic Result

An arithmetic statement can contain one or more resultant identifiers (data items to contain results). In this case, the results of an arithmetic statement are computed as follows:

1. In a statement, all data items to be initially evaluated are computed as required. The result is stored in a temporary data item.
2. The temporary data item obtained in (1) is computed for each resultant identifier, and the results are stored. This computation is done in the order in which the identifiers are specified (from left to right).

An example of obtaining more than one result is shown below. temp indicates a temporary storage field created by the compiler.

Example

Example 1

"ADD a b c TO c d(c) e" is computed as follows:

```
ADD a b c GIVING temp
ADD temp TO c
ADD temp TO d(c) *>...  The value of c was changed by the preceding addition.
ADD temp TO e
```

Example 2

"MULTIPLY a(i) BY i a(i)" is computed as follows:

```
MOVE a(i) TO temp
MULTIPLY temp BY i
MULTIPLY temp BY a(i)
```

6.3.8 ROUNDED Phrase

A ROUNDED phrase can be specified in arithmetic statements.

The number of the decimal places obtained by an arithmetic operation may be greater than that of a resultant identifier. The following processing is performed depending on whether the ROUNDED phrase is present:

1. If the ROUNDED phrase is not specified, the fraction part obtained by the arithmetic operation is rounded down to the length of the resultant identifier.
2. If the ROUNDED phrase is specified, the absolute value of the resultant identifier is increased by one whenever the most significant digit of the fractional part of the result is greater than or equal to five.

The low-order integer position of the resultant identifier may specify the PICTURE symbol P. If so, truncation or rounding down of the low-order integer position occurs in the storage allocated.

Note

When the result of an arithmetic operation is floating-point data and the resultant identifier is a fixed point data item, the rounded value is transcribed to the resultant identifier whether or not a ROUNDED phrase has been specified (Refer to

"Rules for Moving (Transcribing) Data"). In this case, whether a ROUNDED phrase is specified or not, does not make any difference to the result.

6.3.9 ON SIZE ERROR Phrase

Execution of an arithmetic statement may cause a size error condition, which can be detected if the ON SIZE ERROR phrase is specified in an arithmetic statement.

Conditions Where a Size Error Condition Occurs

A size error condition occurs under any of the following conditions:

1. The base of exponentiation is zero, and the exponent is zero or less.
2. The result of evaluating exponentiation is not a real number.
3. The divisor is zero.
4. The absolute value obtained by an arithmetic operation exceeds the maximum value that can be stored in a resultant identifier or is not in the range defined for a BINARY-CHAR/SHORT/LONG/DOUBLE data item in the table in the "USAGE Clause" section of Chapter 5.

The maximum value that can be stored in a resultant identifier in (4) is specified by the PICTURE character-string. If a resultant identifier is a binary data item, this maximum value is also determined by the PICTURE character-string; it is not the maximum value that can be stored in storage.

A size error condition (4) only occurs when the final result is stored by one arithmetic operation. It does not occur when an intermediate result is stored. If the ROUNDED phrase is specified, a size error condition in (4) is checked after rounding off.

If two or more resultant identifiers are specified, a size error condition is checked each time the result of the corresponding arithmetic operation is obtained.

Operation Performed if a Size Error Condition Occurs

The value of a resultant identifier in which a size error condition occurred is as follows:

1. With the ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase, the value of an identifier in which a size error condition occurred remains unchanged. (The value before execution of an arithmetic statement.)
2. Without the ON SIZE ERROR phrase and NOT ON SIZE ERROR phrase, the value of an identifier in which a size error condition occurred is undefined.
3. In an identifier in which no size error condition occurred, the result of the arithmetic operation is stored. This is done regardless of whether an ON SIZE ERROR phrase and a NOT ON SIZE ERROR phrase are written.

After completion of the arithmetic operation, that is, after all the values of resultant identifiers have been determined, control is transferred according to the following rules:

1. Control is transferred to an imperative statement in which the ON SIZE ERROR phrase is written. After the statement is executed, control is transferred to the end of the arithmetic statement. However, the imperative statement may specify a procedure branching statement or a conditional statement that causes an explicit transfer of control. If so, control is transferred according to the rules of the statement.
2. Without the ON SIZE ERROR phrase, control is transferred to the end of the arithmetic statement. It terminates abnormally when the divisor is zero.

Operation Performed if No Size Error Condition Occurs

If no size error condition occurs, after completion of an arithmetic operation, control is transferred according to the following rules:

1. Control is transferred to an imperative statement in which NOT ON SIZE ERROR phrase is written. After the statement is executed, control is transferred to the end of the arithmetic statement. The imperative statement may

specify a procedure branching statement or a conditional statement that causes an explicit transfer of control. If so, control is transferred according to the rules of the statement.

2. Without NOT ON SIZE ERROR, control is transferred to the end of the arithmetic statement.

6.3.10 CORRESPONDING Phrase

In the MOVE, ADD, and SUBTRACT statements, a CORRESPONDING phrase can be specified. The CORRESPONDING phrase associates data items having the same name but belonging to different group items with each other.

When a CORRESPONDING phrase is specified in the following format:

CORRESPONDING d1 TO d2

D1, d2 and the data items to be associated with must satisfy the following conditions:

1. D1 and d2 must be group items. Data items having a level number of 66, 77, or 88 cannot be specified as d1 and d2.
2. A data item containing the USAGE IS INDEX clause cannot be specified as d1 or d2.
3. D1 and d2 must not be reference modified.
4. The names of the data items to be associated must be unique by appending implicit qualifiers to them.
5. When used with the MOVE statement, at least one of the data items to be associated must be an elementary item. The combination of the data items to be associated must conform to the rules for transcribing data.
6. When used with the ADD or SUBTRACT statements, the data items to be associated must be numeric data items.

If "CORRESPONDING d1 TO d2" is specified, data items belonging to d1 are associated with those belonging to d2. The data items contained within d1 and d2 must satisfy all the following conditions:

1. The data-names within d1 and d2 are identical, not designated by the keyword FILLER, and are identically qualified.
2. If the data items subordinate to d1 and d2 specify the REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clauses, they are ignored. Further, data items specifying the REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clauses that are subordinate to the data items specifying the REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clauses are ignored.

6.3.11 Overlapping of Operands

Data items defined in different data description entries may be specified as the sending and receiving fields in one statement. Also, the data items may partially or entirely share the same storage. If so, the result of executing the statement is undefined. Data items defined in the same data description entry may be specified as the sending and receiving fields in one statement. If so, the result of executing the statement is undefined. The rules governing this behavior are defined in the section "Common Statement Rules" earlier in this chapter.

6.3.12 INVALID KEY Phrase

The DELETE, READ, REWRITE, START, or random access WRITE statements may be executed for relative or indexed files. If so, an invalid key condition may occur. An invalid key condition can be detected by specifying the INVALID KEY phrase with the above input-output statements. The NOT INVALID KEY phrase can check whether an input-output statement was executed successfully without an invalid key condition.

The following topics explain the actions taken under the following conditions:

- When an invalid key condition occurs
- When an exception condition other than an invalid key condition occurs
- When the input-output statement is successfully executed

An Invalid Key Condition

If an invalid key condition occurs, the input-output statement executed unsuccessfully. After the value indicating an invalid key condition is stored in the input-output status associated with the file (if any), control is transferred depending on whether:

- The INVALID KEY phrase is specified in the input-output statement.
- A related USE AFTER STANDARD EXCEPTION procedure is defined.

The table below shows where control is transferred if an invalid key condition occurs:

Whether INVALID KEY Phrase is Specified	Whether the USE AFTER STANDARD EXCEPTION Procedure is Defined	Where Control is Transferred if an Invalid Key Condition Occurs
Specified	Defined or not defined	Control is transferred to the imperative statement specified in the INVALID KEY phrase. After the imperative statement is executed, control is transferred to the end of an input-output statement. (*1)
Not specified	Defined	Control is transferred to the USE AFTER STANDARD EXCEPTION procedure. Control is transferred according to the rules of the USE statement.
Not provided	Not defined	If the associated file contains the FILE STATUS clause, control is transferred to the end of an input- output statement. Otherwise, the execution result is undefined.

*1 : The imperative statement may specify a procedure branching statement or a conditional statement that causes an explicit transfer of control . If so, control is transferred according to the rules of the statement.

An Exception Condition Other Than an Invalid Key Condition

If an exception condition other than an invalid key condition occurs, the input-output statement was executed unsuccessfully. After the value indicating an exception condition is stored in the input-output status associated with the file (if any), control is transferred according to the following rules:

1. If the related USE AFTER STANDARD EXCEPTION procedure is defined, control is transferred to the procedure. Then, control is transferred according to the rules of the USE statement.
2. When there is no related USE AFTER STANDARD EXCEPTION procedure, and if a FILE STATUS is associated with the file, the associated FILE STATUS is updated with the input-output status and control is transferred to the end of the input-output statement. If there is no FILE STATUS clause associated with the file, execution results are undefined.

Input-Output Statement Successfully Executed

If no invalid key condition or other exception condition occurs, the input-output statement was executed successfully. After the value indicating this fact is stored in the associated FILE STATUS data item (if any), control is transferred according to the following rules:

1. If the input-output statement contains a NOT INVALID KEY phrase, control is transferred to an imperative statement associated with the NOT INVALID KEY phrase. After the imperative statement is executed, control is transferred to the end of the input-output statement. However, the imperative statement may specify a procedure branching statement or a conditional statement that causes an explicit transfer of control. If so, control is transferred according to the rules of the statement.
2. If the input-output statement does not contain a NOT INVALID KEY phrase, control is transferred to the end of the input-output statement.

6.3.13 AT END Phrase

A sequential access READ statement may be executed for sequential, relative, or indexed files. If so, an "at end" condition may occur. An "at end" condition can be detected by specifying the AT END phrase with the READ statement. The NOT AT END phrase can check whether an input-output statement was executed successfully without an "at end" condition.

The following topics explain the actions taken under the following conditions:

- When an "at end" condition occurs
- When an exception condition other than an "at end" condition occurs
- When the input-output statement is successfully executed

An "At End" Condition

If "an at" end condition occurs, the READ statement was executed unsuccessfully. After the value indicating an "at end" condition is stored in the input-output status associated with the file (if any), control is transferred depending on whether:

a. The AT END phrase is specified in the READ statement.

b. There is a related USE AFTER STANDARD EXCEPTION procedure.

The following table shows where control is transferred if an "at end" condition occurs:

Whether AT END Phrase is Specified	Whether the USE AFTER STANDARD EXCEPTION Procedure is Defined	Where Control is Transferred if an At End Condition Occurs
Specified	Defined or not defined	Control is transferred to the imperative statement specified in the AT END phrase. After the imperative statement is executed, control is transferred to the end of the READ statement. (*1)
Not specified	Defined	Control is transferred to the USE AFTER STANDARD EXCEPTION procedure. After the USE AFTER STANDARD EXCEPTION procedure is executed, control is transferred to the end of the READ statement.
Not specified	Not defined	If the file contains the FILE STATUS clause, control is transferred to the end of the READ statement. Otherwise, the execution result is undefined.

*1: The imperative statement may specify a procedure branching statement or a conditional statement that causes an explicit transfer of control. If so, control is transferred according to the rules of the statement.

An Exception Condition Other Than an At End Condition

If an exception condition other than an "at end" condition occurs, the READ statement was executed unsuccessfully. After the value indicating an exception condition is stored in the input-output status associated with the file (if any), control is transferred according to the following rules:

1. If a related USE AFTER STANDARD EXCEPTION procedure is defined, control is transferred to the procedure. Then, control is transferred according to the rules of the USE statement.

F

2. If there is no USE AFTER STANDARD EXCEPTION procedure associated with the file, and there is FILE STATUS clause associated with the file, control is transferred to the end of the READ statement. If the FILE STATUS clause is not specified, the execution result is undefined.

Input-Output Statement Successfully Executed

If no "at end" condition or other exception condition occurs, the READ statement was executed successfully. After the value indicating the fact is stored in the input-output status associated with the file (if any), control is transferred according to the following rules:

1. If the READ statement contains a NOT AT END phrase, control is transferred to the imperative statement specified in the NOT AT END phrase. After the imperative statement is executed, control is transferred to the end of the READ statement. However, the imperative statement may specify a procedure branching statement or a conditional statement that causes an explicit transfer of control. If so, control is transferred according to the rules of the statement.

2. If the READ statement does not contain the NOT AT END phrase, control is transferred to the end of the READ statement.

6.3.14 Incompatible Data

The reference result of the data item in procedure division is undefined when the content of the data item is incompatible with the class of the data item by the PICTURE clause or the class by the function, except when the data item is referred by the class condition. The internal representation of a numeric data item, refer to the topic "USAGE clause" in the "Data Description Entry" section of Chapter 5, the "Data Division".

6.4 Statements

This section explains each statement that can be specified in the Procedure Division.

6.4.1 ACCEPT Statement (Nucleus)

The ACCEPT statement causes data keyed at the console or supplied by the operating system to be made available for the program in the specified data item.

Format 1

To enter data:

```
ACCEPT identifier-1 {[FROM mnemonic-name-1]}
```

Format 2

To obtain date, day of the week, and time:

```
ACCEPT identifier-2 FROM { DATE
                         { DAY
                         { DAY-OF-WEEK
                         { TIME
```

Syntax Rules

1. Identifier-1 must be
   - alphabetic data item,
   - alphanumeric data item,
   - zoned decimal data item,
   - packed decimal data item,
   - binary data item,
   - external Boolean data item,
   - or fixed-length group item

2. Associate mnemonic-name-1 with the function-name CONSOLE or SYSIN in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. For details on the maximum length of identifier-1 if CONSOLE is specified as mnemonic-name-1, see Appendix B, "Quantitative System Restrictions".

3. Identifier-2 must be an alphanumeric data item, alphanumeric edited data item, numeric edited data item, zoned decimal data item, packed decimal data item, binary data item, or fixed-length group item.

General Rules

Rules for Format 1

1. The ACCEPT statement transfers data keyed from either of the following hardware devices, and stores it in identifier-1. The data input is not edited, and is not checked for errors.
   - CONSOLE (system logical console)
   - SYSIN (system logical input unit)
2. If the FROM phrase is omitted, the mnemonic-name for SYSIN is assumed.
3. If the input-output device indicated in mnemonic-name-1 is the same as the input device specified in a READ statement, the result is undefined.
4. If mnemonic-name-1 corresponds to CONSOLE, processing is performed in the following order:
   a. A message generated by the system is automatically displayed on the system logical console, and execution of the ACCEPT statement is interrupted.
   b. When the user enters a message on the system logical console, the ACCEPT statement restarts. The message is stored in identifier-1 and left justified regardless of the description of the PICTURE clause. If the input message is shorter than identifier-1, the remaining portion of identifier-1 is padded with blanks. If the input message is longer than identifier-1, the message is truncated on the right to the length of identifier-1.
5. If mnemonic-name-1 corresponds to SYSIN, records are repeatedly read, and sequentially stored in receiving data items. This processing is performed until the receiving data items are filled with input data, or until all the records have been read.

Rules for Format 2

1. The format 2 ACCEPT statement moves information described in the FROM phrase into the data item identifier-2. This is done according to the rules for the MOVE statement. Note: END-ACCEPT can optionally terminate the statement.

   DATE (year, month, and day), DAY (year and day), DAY-OF-WEEK (day of the week), and TIME (time) are virtual data items. These can only be used in the ACCEPT statement, and not in any other COBOL statement.
2. DATE is treated as an unsigned six-digit zoned decimal integer item. If DATE is specified, the last two digits of the current year, the current month, and current day are moved into identifier-2 (A total of six digits). For example, if the date is October 1, 1994, the value moved to identifier-2 is 941001.
3. DAY is treated as an unsigned five-digit zoned decimal integer item. If DAY is specified, the following are moved into identifier-2:
   a. The last two digits of the current year
   b. Then, the total number of the days since January 1 (a total of five digits). For example, if the date is October 1, 1994, the value moved to identifier-2 is 94274.
4. TIME is treated as an unsigned eight-digit zoned decimal integer item. If TIME is specified, the following are moved into identifier-2:
   a. Current hour (24-hour format),
   b. Minute
   c. Second
   d. Hundredths of a second

   (A total of eight digits). For example, if the time is precisely 2:41 p.m., the value moved to identifier-2 is 14410000.
5. DAY-OF-WEEK is treated as an unsigned one-digit zoned decimal integer item. If DAY-OF-WEEK is specified, one digit indicating the current day of the week is copied into identifier-2. If the current day of the week is Monday, 1 is moved. If it is Tuesday, 2 is moved. If it is Sunday, 7 is moved.

F

6.4.2 ACCEPT Statement (Screen Handling)

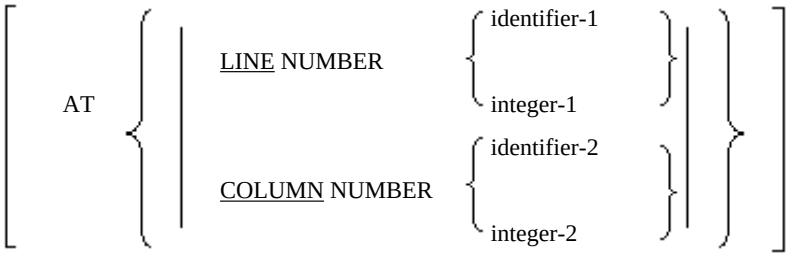
The ACCEPT statement causes data keyed on the screen to be made available for the program in the specified data item.

Note

The screen handling module cannot be used in [.NET], [Linux], [LinuxIPF] or [Linux64].

Format

ACCEPT data-name-1



[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-ACCEPT]

Syntax Rules

1. Data-name-1 must be a screen item defined in the SCREEN SECTION. The screen item must be one of the following:
   - An elementary screen item which the specifies the TO or the USING phrase in the PICTURE clause
   - A group screen item containing an elementary screen item for which specifies the TO or the USING phrase in the PICTURE clause
2. Data-name-1 can be qualified.
3. Integer-1 and integer-2 must be unsigned integers.
4. Identifier-1 and identifier-2 must be unsigned integer items.
5. The LINE NUMBER and COLUMN NUMBER phrases can be specified in any order.

General Rules

1. The ACCEPT statement inputs data from the screen field corresponding to the screen item for data-name-1. Then, the data input is moved into the data item specified in TO or USING phrase of the PICTURE clause in data-name-1. If a group screen item is specified as data-name-1, all the input and update items belonging to the screen item are moved. Data is moved from screen items according to the standard rules for moving data.
2. If a group screen item is specified as data-name-1, the following occurs:
   a. Once the ACCEPT statement has been executed, data can be entered into all the input and update items belonging to the group screen item.
   b. To move among screen items, move the cursor. Movement among screen items is done in the order specified by the LINE NUMBER and COLUMN NUMBER clauses of the screen data description entry.
   c. If the AUTO clause is specified in the screen data description entry of data-name-1, the cursor is automatically moved to the next input or update field when a field is filled.
3. The ACCEPT statement is completed by pressing the input key.

4. If the AUTO clause is specified for a screen item, the ACCEPT statement is completed when:
    a. Data is entered into the field of the last screen item of the group screen item for data-name-1.
    b. Data is entered into the field of the elementary screen item for data-name-1.
5. The LINE NUMBER phrase specifies the screen line number corresponding to the screen item for data-name-1. The first line of the physical screen is considered to be line number 1.
6. The COLUMN NUMBER phrase specifies the screen column number corresponding to the screen item for data-name-1. The first column of the physical screen is considered to be column number 1.
7. If the AT phrase is omitted, data is entered starting at the first line and first column of the screen.
8. If the ON EXCEPTION phrase is specified, and input does not terminate normally, imperative-statement-1 is executed.
9. If the NOT ON EXCEPTION phrase is specified, and input terminates normally, imperative-statement-2 is executed. For information on how to determine the termination status of an ACCEPT statement, refer to the topic "CRT STATUS clause" in the "Configuration Section" section of Chapter 4, the "Environment Division".

F

6.4.3 ACCEPT Statement (Command Line Arguments and Environmental Variables)

The ACCEPT statement inputs a) the number of command line arguments on the command line, or b) the command line argument value, or c) the value of an environmental variable.

Format

ACCEPT identifier-1 FROM mnemonic-name-1

[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-ACCEPT]

Syntax Rules

1. Mnemonic-name-1 must be associated with the following function-name in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION:
    - ARGUMENT-NUMBER
    - ARGUMENT-VALUE
    - ENVIRONMENT-VALUE
2. Identifier-1 must be a data item that satisfies the following conditions:
    a. Mnemonic-name-1 corresponds to the function-name ARGUMENT-NUMBER. If so, identifier-1 must be an unsigned integer item.
    b. Mnemonic-name-1 corresponds to the function-name ARGUMENT-VALUE or ENVIRONMENT-VALUE. If so, identifier-1 must be a group item that does not contain a variable occurrence data item or an alphanumeric data item.
3. Mnemonic-name-1 may correspond to the function-name ARGUMENT-NUMBER. If so, the ON EXCEPTION and NOT ON EXCEPTION phrases must not be specified.

General Rules

- If Mnemonic-name-1 is Associated with the Function-name ARGUMENT-NUMBER

  The number of the arguments specified on the command line is moved into identifier-1 according to the standard rules for the movement of data.

- If Mnemonic-name-1 is Associated with the Function-name ARGUMENT-VALUE

    1. The value of the command line argument indicated by the current argument position indicator is moved into identifier-1 according to the standard rules for the movement of data.
    2. Execution of the ACCEPT statement increments the value of the argument position indicator by one.
    3. The value of the argument indicator before execution of the ACCEPT statement may have exceeded the position of the last argument on the command line. If so, and the ON EXCEPTION phrase is specified, imperative-statement-1 is executed.
    4. If the exception condition in (3) does not occur and the NOT ON EXCEPTION phrase is specified, imperative-statement-2 is executed.

- If Mnemonic-name-1 is the Function-name ENVIRONMENT-VALUE

    1. Before execution of the ACCEPT statement, a DISPLAY statement specifying a mnemonic-name corresponding to the function-name ENVIRONMENT-NAME must be executed. Execution of the ACCEPT statement moves the value of the environment variable positioned by the preceding DISPLAY statement into identifier-1. This is done according to the standard rules for the movement of data.
    2. If the ON EXCEPTION phrase is specified, imperative-statement-1 is executed under either of the following conditions:

        a. The DISPLAY statement for positioning the environmental variable is not executed prior to execution of the ACCEPT statement.

        b. The environmental variable specified in the preceding DISPLAY statement does not exist.

    3. If the exception condition in (2) does not occur and the NOT ON EXCEPTION phrase is specified, imperative-statement-2 is executed.

6.4.4 ADD Statement (Nucleus)

The ADD statement performs an addition operation.

Format 1

To replace the result of addition with the addend:

```
ADD { identifier-1 } ... TO {identifier-2 [ROUNDED]} ...
    { integer-1    }
    [ON SIZE ERROR imperative-statement-1]
    [NOT ON SIZE ERROR imperative-statement-2]
    [END-ADD]
```

Format 2

To store the result of addition in a data item different from the addend:

```
ADD { identifier-1 } ...TO { identifier-2 } GIVING
    { integer-1    }       { integer-2    }
    {identifier-3 [ROUNDED]} ...
    [ON SIZE ERROR imperative-statement-1]
    [NOT ON SIZE ERROR imperative-statement-2]
    [END-ADD]
```

Format 3

To add the corresponding data items belonging to two group items:

```
ADD { CORRESPONDING } identifier-1 TO identifier-2
    { CORR         }
    [ROUNDED]
    [ON SIZE ERROR imperative-statement-1]
    [NOT ON SIZE ERROR imperative-statement-2]
    [END-ADD]
```

Syntax Rules

1. In formats 1 and 2, identifier-1 and identifier-2 must be numeric data items. In format 3, identifier-1 and identifier-2 must be group items.
2. Identifier-3 must be a numeric data item or a numeric edited data item.
3. Literal-1 and literal-2 must be numeric literals.
4. CORRESPONDING is synonymous with CORR. Either can be specified.

General Rules

Rules for Format 1

The ADD statement sums the individual operands, in the order specified, before the TO keyword to identifier-2, and stores the result in identifier-2.

Rules for Format 2

The ADD statement sums the individual operands, in the order specified, before the TO keyword to the operand after the TO keyword, and stores the result in identifier-3.

Rules for Format 3

The ADD statement obtains the sum of subordinate data items in identifier-1 and the corresponding subordinate data items in identifier-2. The subordinate data items in identifier-1 and identifier-2 to be summed must have the same names and the same qualifiers. The subordinate data items in identifier-1 and identifier-2 are treated as the addends. The sum is stored in the subordinate data items of identifier-2. The result is equivalent to adding each subordinate data item with the same names and qualifiers of identifier-1 and identifier-2 individually.

Rules Common to All Formats

1. The END-ADD phrase delimits the scope of the ADD statement.
2. For details of the ROUNDED, ON SIZE ERROR, and CORRESPONDING phrases, and the rules for movement of data, see the section titled "Common Statement Rules" earlier in this chapter.

6.4.5 ALTER Statement (Nucleus)

The ALTER statement modifies the predefined processing order. It is an obsolete language element and its use is very strongly discouraged.

Note

The ALTER statement cannot be used in [.NET].

Format

ALTER { procedure-name-1 TO [PROCEED TO] procedure-name-2 }...

Syntax Rules

1. Procedure-name-1 must be a paragraph name. The paragraph must consist of only one sentence containing one GO TO statement without the DEPENDING phrase.
2. Procedure-name-2 must be a paragraph or section-name in the PROCEDURE DIVISION.

General Rules

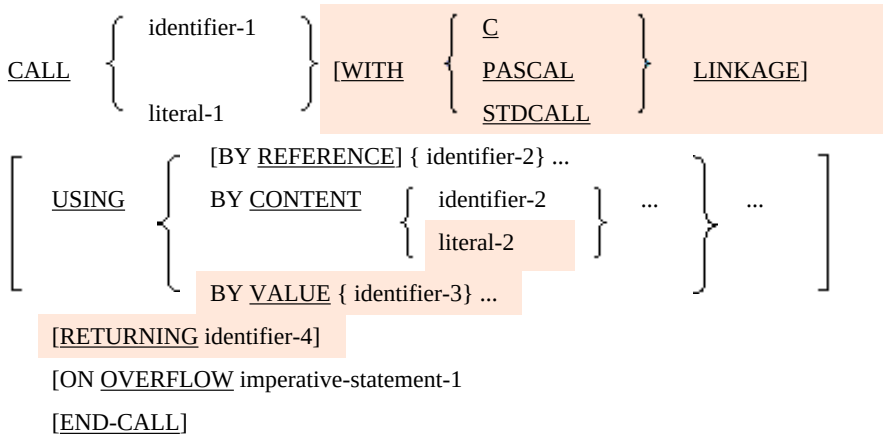
The ALTER statement changes the destination of the GO TO statement specified in the paragraph procedure-name-1 to procedure-name-2.

6.4.6 CALL Statement (Inter-program Communication)

The CALL statement transfers control to another program in the run unit.

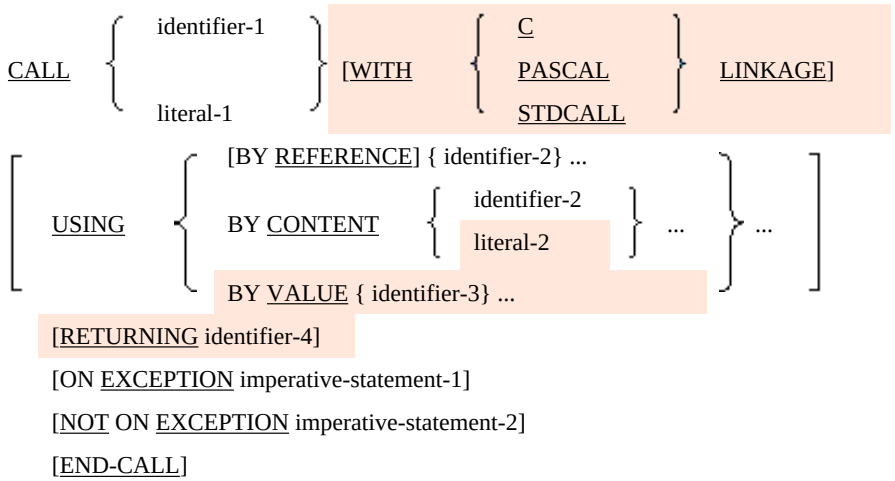
Format 1

ON OVERFLOW phrase



Format 2

ON EXCEPTION phrase



F

[Win16] [Win32] The WITH phrase can be specified.

The RETURNING phrase is available for [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

Syntax Rules

1. Identifier-1 must be an alphanumeric data item.

F

   Identifier-1 can specify National data items in [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

2. Identifier-2, identifier-3 and identifier-4(*) must be data items defined in the following:
   - FILE SECTION
   - WORKING-STORAGE SECTION
   - LINKAGE SECTION
   - LOCAL-STORAGE SECTION ([Winx64], [Linux64] and [.NET] only)
3. Identifier-2 and identifier-4(*) must be data items with level-numbers 01 or 77, or elementary items at any level.

F

   However, in this implementation of COBOL, they may be group items at any level.

F

4. If identifier-2 or identifier-4 specify an internal Boolean data item, they must be defined so that they start on a byte boundary.
5. Identifier-3 must be one of the following:
   - A nine digit or less numeric data item with a USAGE clause specifying COMPUTATIONAL-5
   - A one character alphabetic data item
   - A one character alphanumeric data item
   - A one digit zoned decimal integer item without SEPARATE
   - In [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET], a numeric data item with a USAGE clause specifying BINARY-CHAR, BINARY-SHORT, or BINARY-LONG.
   - In [Winx64], [LinuxIPF] and [Linux64], an eighteen digit or less numeric data item with a USAGE clause specifying COMPUTATIONAL-5, a numeric data item with a USAGE clause specifying BINARY-DOUBLE.
6. In [Winx64], [LinuxIPF] and [Linux64], the following items cannot be specified for identifier-4:
   - A group item in which COMPUTATIONAL-1 of a USAGE clause is the only subordinate elementary item
   - A group item in which COMPUTATIONAL-2 of a USAGE clause is the only subordinate elementary item
7. LIteral-1 must be a nonnumeric literal.

F

   Identifier-1 can specify a national language nonnumeric literal in [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

F

8. Literal-2 must be a nonnumeric literal, hexadecimal nonnumeric literal, or national nonnumeric literal.
9. For details of the maximum number of operands that can be specified in the USING phrase, see Appendix B, "Quantitative System Restrictions".

F

* : Extended functions or functions specific to NetCOBOL.

- identifier-4

General Rules

1. The CALL statement calls and transfers control to another program in the run unit. The program containing the CALL statement is referred to as the calling program. The program called by execution of the CALL statement is referred to as the called program. The name of the called program is specified in identifier-1 or literal-1. To pass parameters to a called program, specify the USING phrase. To check whether a called program can be executed, specify the ON OVERFLOW, ON EXCEPTION, or NOT ON EXCEPTION phrase.

2. If the called program is included in the same compilation unit as the calling program, literal-1 or identifier-1 must specify the internal name of the called program. If the called program is in a different compilation unit, literal-1 or identifier-1 must specify the external name(*) of the called program.

3. When the CALL statement is executed, if the called program can be executed, control is transferred to the called program.

4. When the CALL statement is executed, where control is transferred depends on whether the ON OVERFLOW, ON EXCEPTION, and NOT ON EXCEPTION phrases are specified. The following table shows where control is transferred when the CALL statement is executed.

Whether the ON EXCEPTION or ON OVERFLOW phrase is specified	Whether the NOT ON EXCEPTION phrase is specified	Operation of the CALL statement	
		If the called program cannot be executed	If the called program can be executed
Specified	Specified	1. Control is transferred to imperative-statement-1. 2. After imperative-statement-1 is executed, control is transferred to the end of the CALL statement. (*1)	Control is transferred to the called program. After returning from the called program, control is transferred to imperative-statement-2. After imperative-statement-2 is executed, control is transferred to the end of the CALL statement. (*2)
Specified	Not specified	1. Control is transferred to imperative-statement-1. 2. After imperative-statement-1 is executed, control is transferred to the end of the CALL statement. (*1)	Control is transferred to the called program. After returning from the called program, control is transferred to the end of the CALL statement.
Not specified	Specified	Operation of the CALL statement is undefined.	Control is transferred to the called program. After returning from the called program, control is transferred to imperative-statement-2. After imperative- statement-2 is executed, control is transferred to the end of the CALL statement. (*2)
Not specified	Not specified	Operation of the CALL statement is undefined.	Control is transferred to the called program. After returning from the called program, control is transferred to the end of the CALL statement.

*1 imperative-statement-1 may specify a procedure branching statement or a conditional statement that causes an explicit transfer of control. If so, control is transferred according to the rules for the statement.

*2 imperative-statement-2 may specify a procedure branching statement or a conditional statement that causes an explicit transfer of control. If so, control is transferred according to the rules for the statement.

5. Program names can be duplicated within a run unit. If the program name specified in the CALL statement is duplicated within a run unit, the program to be called is determined according to the rules for the scope of names as follows.

    a. The program names directly included in the program containing the CALL statement are checked. If one of these programs is specified in the CALL statement, the program is called.

b. If the program with the duplicate name specified in the CALL statement possesses the COMMON attribute, and is directly contained within another program that directly or indirectly contains the program specifying the CALL statement, that common program is called unless the calling program is contained in that common program.

c. Another compilation unit is checked to see if it contains the program specified in the CALL statement. If such a program is found, the program is called.

6. If the called program does not possesses the INITIAL attribute, the program, and all the programs directly or indirectly contained in it are initialized under the following conditions:

   a. When they are called for the first time in the run unit.

   b. When they are called for the first time after the CANCEL statement is executed for the called program.

   c. Otherwise, the following are left as they were when control is returned to the calling program:

      - program(s) called for the second and subsequent times in the run unit
      - program(s) directly or indirectly included in such program(s)

7. If the called program has the INITIAL attribute, the program and programs contained therein, directly or indirectly, are initialized each time they are called.

8. If the called program is initialized, the internal file connectors of the called program are closed. Otherwise, the status and positioning of files for the internal file connectors will have not been changed since the program was last executed.

   For details of the initial program status, see the topic titled "Initial State of a Program" in the "Inter-program Communication Module" section of Chapter 2, "COBOL Modules".

9. The status and positioning of files using external file connectors are unaffected when calling and returning from programs.

10. The USING phrase can be specified in a CALL statement only if the USING phrase is specified in the called COBOL program in either of the following:

    a. The PROCEDURE DIVISION header

F   b. An ENTRY statement

    In either case, the number of the operands in the USING phrase of the calling program must equal the number of operands in the called program.

11. The operands of the USING phrase in the CALL statement specify the parameters to be passed to the called program. If the called program is a COBOL program, it receives the parameters via the PROCEDURE DIVISION header or the ENTRY statement USING phrase. Parameters in the calling program and called program are associated with other by the order in which they are specified in the USING phrases. For example, to pass two parameters to a COBOL program, specify them in the USING phrase of the CALL statement. The first and second parameters of the CALL statement are associated with the first and second parameters in the PROCEDURE DIVISION header (or the ENTRY statement(*)) of the called program. The parameters in the called program must be defined in the LINKAGE SECTION, and the data description entries must exactly match the length and usage as defined in the calling program.

12. The values of the parameters in the USING phrase of the CALL statement can be used in the called program. The parameter values are available for the called program when the CALL statement is executed.

13. Each parameter can specify the BY CONTENT, BY REFERENCE, or BY VALUE phrase(*). When a parameter specifies one of these phrases, that phase is applied to all subsequent parameters that omit the BY CONTENT, BY REFERENCE, or BY VALUE phrases until another BY CONTENT, BY REFERENCE, or BY VALUE phrase is encountered. If the first parameter in a CALL statement is not preceded by the BY CONTENT, BY REFERENCE, or BY VALUE phrase, the BY REFERENCE phrase is assumed.

14. The BY REFERENCE phrase may be specified explicitly or implicitly. If this phrase is specified, the parameters for the calling and called program occupy the same storage location in memory. When this phrase is specified, the called program may modify the parameter. The modified value is reflected in the corresponding parameter in the calling program.

15. If the BY CONTENT phrase is specified, copies of the values of the parameters specified in the calling program are passed to the called program. The called program can reference the copies of the parameters. If, however, the called program modifies these parameters, the values are not reflected in the corresponding parameters of the calling program.

F

16. Literal-2 may be specified in the BY CONTENT phrase. If so, the corresponding data item attribute and character position count for the called program must be the same as those for literal-2.

17. If the BY VALUE phrase is specified, the values of the parameters are passed directly to the called program. The called program must be written in a language (such as C) that enables parameter values to be received. The called program may modify the passed values. If so, the values of the corresponding data items for the calling program are not changed.

18. The called program cannot execute a CALL statement that directly or indirectly calls the calling program.

19. The CALL statement must not directly or indirectly call an incomplete program, that is, .a program that has already been called and has not completed execution.

20. The END-CALL phrase delimits the scope of the CALL statement.

21. To transfer control to the beginning of the PROCEDURE DIVISION of the called program, specify the name of the called program as literal-1 or identifier-1.

F

It is possible to transfer control to an entry point other than the beginning of the PROCEDURE DIVISION of the called program. This is accomplished by specifying the secondary entry name specified in the ENTRY statement as literal-1 or identifier-1.

22. When you intend to call a program that can reference a GLOBAL name, do not specify the GLOBAL name in the USING phrase of the CALL statement.

23. Identifier-1, identifier-2, and identifier-4(*) are evaluated prior to execution of the CALL statement.

F

24. If the RETURNING phrase is specified, the result of the program identified by identifier-1 or literal-1 is stored in identifier-4.

25. If the RETURNING phrase is specified, the PROCEDURE DIVISION header of the called program must specify the RETURNING phrase.

26. If the RETURNING phrase is not specified, the PROCEDURE DIVISION header of the called program should not specify the RETURNING phrase.

27. If the called program moves a value to the data item specified in the RETURNING phrase, that value will be available in the data item specified in the RETURNING phrase of the CALL statement.

28. The usage and length of a data item specified in the RETURNING phrase must be the same for the program that calls and the program that is called.

F

* : Extended functions or functions specific to NetCOBOL.

- external name
- ENTRY statement
- BY VALUE phrase
- identifier-4

F

Rules for the WITH Phrase

1. The WITH phrase indicates that the object program is called according to the specified calling conventions. If the WITH phrase is omitted, the COBOL calling convention is applied. These calling conventions include:
   - COBOL calling convention
   - C calling convention (WITH C LINKAGE)
   - PASCAL calling convention (WITH PASCAL LINKAGE)

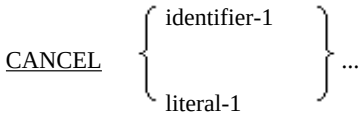
- STDCALL calling convention (WITH STDCALL LINKAGE)

2. The calling conventions between the calling and called programs must be identical. If they are different, the execution result is undefined.
3. The WITH phrase must not be used in calling a nested program.

6.4.7 CANCEL Statement (Inter-program Communication)

The CANCEL statement removes a subprogram from memory. After a program is canceled, the next time it is loaded into memory via a CALL statement, it will be reset to its initial state.

Format



Syntax Rules

1. Identifier-1 must be an alphanumeric data item.

   F

   Identifier-1 may, however, specify a national data item in [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

2. Literal-1 must be a nonnumeric literal.

   F

   Literal-1 may specify a national language nonnumeric literal in [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

General Rules

1. The CANCEL statement cancels the logical relationship between a run unit and a subprogram. The name of the program to be canceled is specified in identifier-1 or literal-1. Execution of the CANCEL statement cancels the logical relation between the following:
   a. The subprogram specified in identifier-1 or literal-1
   b. The run unit of the program containing the CANCEL statement
2. If a canceled program is in the same compilation unit as the program that specified the CANCEL statement, the contents of identifier-1 and literal-1 data item must specify the internal name of the program to cancel. Otherwise, identifier-1 or literal-1 must specify the external name(*) of the program to cancel.
3. Execution of the CANCEL statement cancels the program specified in the CANCEL statement, and all the nested programs included in the canceled program.
4. After a CANCEL statement, or an implicit CANCEL statement, has executed successfully, the program specified in the CANCEL statement may be called again by the run unit. If so, the called program is re-initialized.
5. The program specified in the CANCEL statement must not directly or indirectly reference the called program in which the EXIT PROGRAM statement has not been executed. For example, CANCEL"A" cannot be executed in a program called during execution of program A.
6. The logical relationship between the run unit and the program canceled is reestablished when the canceled program is again called.
7. The program called by the CALL statement is canceled under any of the following conditions:
   a. A CANCEL statement for the program is executed.
   b. The run unit including the program terminates.
   c. The program has the INITIAL attribute, and executes the EXIT PROGRAM statement.

In cases (b) and (c), the CANCEL statement is called an implicit CANCEL statement.

F

* : Extended functions or functions specific to NetCOBOL.

- external name

6.4.8 CLOSE Statement (Sequential, Relative, Indexed, Presentation File(*), and Report Writer Module)

The CLOSE statement terminates file processing.

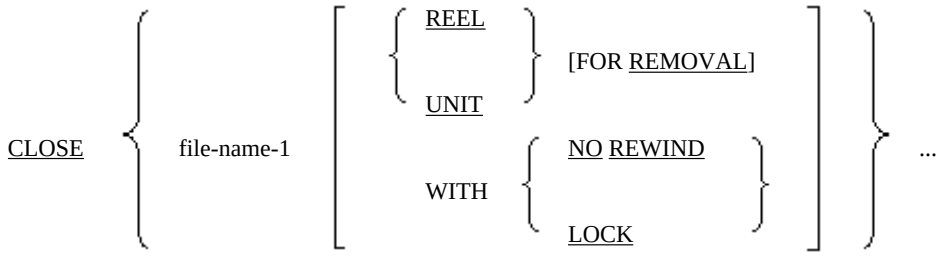
F

* : Extended functions or functions specific to NetCOBOL.

- Presentation File

Format 1

Sequential and report files



Note

The report writer module cannot be used in [Winx64], [LinuxIPF] and [.NET].

Format 2

Relative, indexed, and presentation file(*)

CLOSE {file-name-1 [WITH LOCK] } ...

Syntax Rules

Rules Common to Format 1 and Format 2

In a list of file-name-1, a file with different ORGANIZATION or ACCESS MODE clauses can be specified.

F

Rules for Format 1

1. If the WITH NO REWIND phrase is specified, file-name-1 must be a record sequential file.
2. If the REEL or UNIT phrase is specified, file-name-1 must be a record sequential or a print file without the FORMAT clause.

General Rules

Operation of the CLOSE statement depends on the storage medium. Operation of the CLOSE statement is explained under the following three headings:

- Unit record (sequential, relative, indexed, presentation(*), and report files)

  For files on an input-output medium that does not support the concepts of rewinding or volumes, these clauses are meaningless.

- Sequential single volume (sequential and report files)

  Sequential file included entirely in one volume

- Sequential multi-volume (sequential and report files)

  Sequential file spanning two or more volumes

Rules Common to Individual Files

1. Execution of the CLOSE statement updates the input-output status of file-name_1.
2. The CLOSE statement in which two or more file-name-1 are specified may be executed. If so, this produces the same result as executing separate CLOSE statements, in the sequence specified, for each file file-name-1.

Rules Common to Relative, Indexed, Presentation(*), Sequential and Report Files for Unit Records

1. The file for file-name-1 must be opened before execution of the CLOSE statement.
2. If you execute a CLOSE statement that does not contain REEL/UNIT phrase, the following processing is executed for file-name-1 files.
   a. Closing of files

      The standard termination procedure is executed.

   b. Locking of files

      If the LOCK phrase is specified, a file is locked, and cannot be reopened in the run unit.
3. If the execution of the CLOSE statement that does not contain REEL/UNIT phrase is successful, the state of file-name-1 files is as follows:
   a. The record fields of file-name-1 cannot be referenced.
   b. All the records retained by the file connector for file-name-1 and file-name-1, itself, are unlocked.
   c. File-name-1 is closed, and is disassociated with its file connector.
4. If the record fields are referenced after the CLOSE statement has executed successfully, the result is undefined.
5. For sequential, relative, and indexed files, an undefined file opened in the input mode may not exist. If so, the file is not created, and the file position indicator remains unchanged.
6. Even if REEL/UNIT phrase contains a CLOSE statement, the file remains open, and nothing happens except the behavior explained in rule1 of "Rules Common to Individual Files".

Rules for Sequential and Report Files for Volumes

1. The following table lists the results of executing the CLOSE statement for volume files.

Method of Writing the CLOSE Statement	Sequential Single Volume Unit	Sequential Multi- Volume Unit
CLOSE	(c), (g)	(a), (c), (g)
CLOSE WITH LOCK	(c), (e), (g)	(a), (c), (e), (g)
CLOSE WITH NO REWIND	(b), (c)	(a), (b), (c)
CLOSE REEL/UNIT	(f), (g)	(f), (g)
CLOSE REEL/UNIT FOR REMOVAL	(d), (f), (g)	(d), (f), (g)

The parenthesized alphabetic characters in the table correspond to the following. If execution results depend on the open mode, there is an explanation for each open mode. Otherwise, the explanation applies to the files opened in any open mode.

a. Influence on the preceding volume

- If a file is opened in the input or input-output mode, all the previous volumes are closed except when the file was previously closed by a CLOSE statement with the REEL/UNIT phrase. If the current volume is not the last volume of the file, the subsequent volumes are not processed.

- If a file is opened in the output mode, all the previous volumes are closed except when the file was previously processed by a CLOSE statement with the REEL/UNIT phrase.

b. The current volume is not rewound.

The current volume is not rewound, but left in the same position.

c. Closing of files

- If a file is opened in the input or input-output mode and the file is in the termination position, and a label record is specified, the label is processed based on the standard label procedure. Then, the standard termination procedure is executed.

Under either of the following conditions, only the standard termination procedure is executed, and termination label processing is not performed:

- The file is in the termination position, and no label record is specified

- The file is not in the termination position.

- If a file is opened in output mode and a label record is specified, the label is processed based on the standard label procedure. Then, the standard termination procedure is executed. If no label record is specified, the standard termination procedure is executed without processing a label.

- If a file is opened in extend mode the standard termination procedure is executed.

d. Removal of volumes

The current volume is rewound, if possible, and logically removed from this run unit. To process the volume again in the proper file order, execute the CLOSE statement without the REEL/UNIT phrase. Then, execute the OPEN statement.

F

In this implementation of COBOL, the FOR REMOVAL phrase is treated as a comment. Note: The FOR REMOVAL phrase cannot be used in [.NET].

e. Locking of files

A file is locked, and cannot be reopened in the run unit.

f. Closing of volumes

If a file is opened in the input or input-output mode:

- The current volume may be the last volume or the only volume of the file. Alternatively, the reel may be on a unit record storage medium. If so, the volume is not replaced, and the volume indicator remains unchanged.

- Another volume of the file, if any, is replaced, and the volume indicator is updated so that it indicates the next volume of the file. Then, the standard start volume label procedure is executed. If the current volume does not contain a data record, the volume is replaced again.

If a file (volume storage medium) is opened in the output mode:

- The standard termination volume label procedure is executed.

- The volume is replaced. The volume indicator is updated so that it indicates a new volume.

- The standard start volume label procedure is executed.

- The next WRITE statement outputs a record in the next position of the next volume.

g. Rewinding

The current volume is physically positioned at the beginning.

F

2. Execution of the CLOSE statement unlocks the file and all records retained by the file connector of file-name-1.

3. If there is no undefined file opened in the input mode, no file termination and volume procedures are executed for the file. The file position indicator and the volume indicator remain unchanged.
4. If the REEL and UNIT phrases are omitted, once the CLOSE statement has executed successfully, the file for file-name-1 has the following status:
    a. The record field for file-name-1 cannot be referenced. If the record field is referenced after the CLOSE statement has executed successfully, the result is undefined.
    b. File-name-1 is closed, and it is disassociated with its file connector.

Rules for Report Files

Before execution of the CLOSE statement, all reports associated with the report file that have been initiated must be terminated by executing the TERMINATE statement.

6.4.9 COMPUTE Statement (Nucleus)

The COMPUTE statement evaluates

- arithmetic

F - or Boolean expression

For details about how to use the examples, refer to "COMPUTE Statement" in the "Syntax Samples".

Format 1

To evaluate an arithmetic expression:

```
COMPUTE {identifier-1 [ROUNDED]} ... = arithmetic-expression-1
    [ON SIZE ERROR imperative-statement-1]
    [NOT ON SIZE ERROR imperative-statement-2]
    [END-COMPUTE]
```

F

Format 2

To evaluate a Boolean expression:

```
COMPUTE {identifier-2} ... = Boolean-expression-1
```

Note

"=" is a key word, but not underlined to avoid confusion with other symbols.

Syntax Rules

1. Identifier-1 must be a numeric data item or a numeric edited data item.

F 2. Identifier-2 must be a Boolean data item.

General Rules

Rules for Format 1

1. A format 1 COMPUTE statement evaluates arithmetic-expression-1 and stores the result in identifier-1 in the sequence specified.
2. If only one identifier or literal is specified in arithmetic-expression-1, the value of the identifier or the literal is stored in identifier-1.

F

Rules for Format 2

1. A format 2 COMPUTE statement evaluates Boolean-expression-1 and stores the result in identifier-2 in the sequence specified.
2. If only one identifier or literal is specified in Boolean-expression-1, the value of the identifier or the literal is stored in identifier-2.

Rules Common to Format 1 and Format 2

1. The END-COMPUTE phrase delimits the scope of the COMPUTE statement.
2. See the section titled "Common Statement Rules" of this chapter for details of the following:
   a. ROUNDED and ON SIZE ERROR phrases
   b. Arithmetic and Boolean expressions
   c. Rules for operations and movement of data

6.4.10 CONTINUE Statement (Nucleus)

The CONTINUE statement indicates that there is no executable statement.

Format

CONTINUE

Syntax Rule

The CONTINUE statement can be written anywhere a conditional or imperative statement can be specified.

General Rule

The CONTINUE statement does not affect program execution.

6.4.11 DELETE Statement (Relative and Indexed Files)

The DELETE statement logically removes a record from a mass storage file.

Format

DELETE file-name-1 RECORD
[INVALID KEY imperative-statement-1]
[NOT INVALID KEY imperative-statement-2]
[END-DELETE]

Syntax Rules

1. If file-name-1 is a sequential access file, the INVALID KEY or NOT INVALID KEY phrase cannot be specified.
2. If file-name-1 is a random or dynamic access file and there is no USE AFTER STANDARD EXCEPTION procedure associated with the file, the INVALID KEY phrase must be specified.

F

However, in this implementation of COBOL, the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.

3. For an indexed file, file-name-1 must not be a random access file for which the RECORD KEY clause with the DUPLICATES phrase is specified.

General Rules

Rules Common to Relative and Indexed Files

1. The file for file-name-1 must be a mass storage file.
2. File-name-1 must be opened in the input-output mode before execution of the DELETE statement.
3. If file-name-1 is a sequential access file, a READ statement must be executed before execution of the DELETE statement. If the READ statement is executed successfully, execution of the DELETE statement removes the record read by the READ statement from a file.
4. F An attempt to delete a record locked by another file connector forces the DELETE statement to be executed unsuccessfully, and a value indicating that the record is locked is stored in the input-output status of file-name-1.
5. After the DELETE statement has executed successfully, the applicable record is logically removed from a file, and the record cannot be referenced.
6. Execution of a DELETE statement does not affect the file position indicator.
7. If the DELETE statement is executed, the following are not changed:
    a. The contents of the record field
    b. The contents of the data item specified in DEPENDING ON phrase of the RECORD clause of the file description entry for file-name-1.
8. Execution of the DELETE statement updates the value of the input-output status for file-name-1.
9. F If LOCK MODE IS AUTOMATIC is specified for file-name-1, after the DELETE statement has executed successfully, the existing records are unlocked.
10. If an invalid key condition occurs during execution of the DELETE statement, a value indicating an invalid key condition is stored in the input-output status for file-name-1, and control is transferred. This is done according to the rules described in the topic titled "INVALID KEY Phrase" in the "Common Statement Rules" section of this chapter.
11. If no exception conditions occur during execution of the DELETE statement, after the input-output status for file-name-1 is updated, control is transferred according to the rules described in the topic titled "INVALID KEY Phrase" in the "Common Statement Rules" section of this chapter.
12. The END-DELETE phrase delimits the scope of the DELETE statement.

Rules for Relative Files

If file-name-1 is a random or dynamic access file, the relative record number of the record to be deleted must be stored in the relative key item before execution of the DELETE statement. Execution of the DELETE statement logically removes the record having the relative record number from the file. If the file does not contain a record having the specified relative record number, an invalid key condition occurs, and execution of the DELETE statement is unsuccessful.

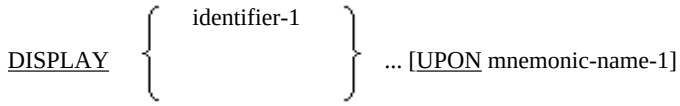
Rules for Indexed Files

If file-name-1 is a random or dynamic access file, the key value of the record to be deleted must be stored in the main record key item before execution of the DELETE statement. Execution of the DELETE statement logically removes the record having the specified key value from the file. If the file does not contain a record having the specified key value, an invalid key condition occurs, and the execution of DELETE statement is unsuccessful.

6.4.12 DISPLAY Statement (Nucleus)

The DISPLAY statement displays small amounts of data.

Format



literal-1

[WITH NO ADVANCING]

Syntax Rules

1. The numeric literal specified in literal-1 must be an unsigned integer.

F

However, in this implementation of COBOL, a numeric literal with a sign or a fractional part can be specified as literal-1.

2. Mnemonic-name-1 must be associated with one of the following function-names in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION:

   - CONSOLE
   - SYSOUT
   - SYSERR
   - SYSPUNCH

General Rules

1. The DISPLAY statement transfers the contents of identifier-1 or literal-1 to a hardware device in the order in which they are specified.
2. If a figurative constant is specified in literal-1, only one repetition of the figurative constant is displayed.
3. If identifier-1 is specified, its contents are displayed in the following format:

   a. A group item or display elementary item (excluding a signed zoned decimal data item without the SEPARATE phrase) may be specified as identifier-1. If so, the content of identifier-1 is displayed as it is.

   b. A binary data item, packed decimal data item, index data item, or signed zoned decimal data item without the SEPARATE phrase may be specified as identifier-1. If so, it is converted to a zoned decimal data item with the SIGN LEADING SEPARATE phrase, and displayed in the following format:

      - If identifier-1 is signed : s9(n) 9(m)
      - If identifier-1 is unsigned : 9(n) 9(m)
      - If identifier-1 is an index data item : 9(n)

      where "s" indicates a sign (+ or -), "9(n)" indicates the number of the digits in the integer part, and "9(m)" indicates the number of digits in the fraction part. For an index data item, the value of n is 9, but in [Winx64], [LinuxIPF] and [Linux64], the value of n is 18.

   c. If a BINARY-CHAR/SHORT/LONG/DOUBLE data item is specified for identifier-1, it is treated like a binary data item with the USAGE COMP-5 clause, and displayed in the following format:

Table 6.1 DISPLAY Representation of Integer Numeric Data Items

USAGE Clause	Maximum Value	Representation (*1)
BINARY-CHAR UNSIGNED	+255	9(3)
INARY-SHORT SIGNED	+32,767	s9(5)
BINARY-LONG SIGNED	+2,147,483,647	s9(10)
BINARY-DOUBLE SIGNED	+9,223,372,036,854,775,807	s9(19)

*1: "s"" indicates a sign (+ or -), "9(n)" indicates the equivalent display numeric item used to display the number.

   d. An internal Boolean data item specified as identifier-1 is converted to an external Boolean data item, and displayed.

e. An internal floating-point data item specified as identifier-1 is converted to an external floating-point data item, and displayed in the following format:

   For a single-precision internal floating-point data item: s.9(8) Es99

   For a long-precision internal floating-point data item: s.9(17) Es99

   where "s" indicates a sign (+ or -), and "9(8)", "9(17)" and "99" indicate the number of digits.

f. A national data item or a national edited data item specified as identifier-1 is displayed as if one of the following were specified as identifier-1:

   - CHARACTER TYPE IS MODE-2 clause without a mnemonic-name
   - CHARACTER TYPE IS MODE-1 clause without a mnemonic-name

g. A pointer data item specified as identifier-1 is displayed in hexadecimal.

4. If two or more operands are specified in a DISPLAY statement, the total byte count (after conversion, if conversion is necessary) of all operands is the transfer byte count.
5. Execution of the DISPLAY statement transfers and displays the contents of identifier-1 or literal-1, in the order in which they are written, as shown below.

   a. If the transfer byte count is the same as the number of bytes that can be transferred at one time, data is transferred and displayed as is.

   b. If the transfer byte count is greater than the number of bytes that can be transferred at one time, data is repeatedly transferred and displayed. Each repetition consists of aligning the left end of data to the leftmost position, then transferring the data. When the last data is transferred, the rule in (a) or (c) is applied.

   c. If the transfer byte count is less than the number of bytes that can be transferred at one time, the left end of data is aligned at the leftmost position, and the data is transferred and displayed. If the device does not accept variable-length records, the remaining portion is padded with spaces.

6. If the UPON phrase is omitted, the mnemonic-name corresponding to SYSOUT is assumed.
7. The WITH NO ADVANCING phrase may be omitted. If so, after the last operand has been transferred to the hardware device, the cursor is positioned to the beginning of the next line.
8. If the WITH NO ADVANCING phrase is specified, after the last operand has been displayed, a line feed is not performed. At this point, the hardware device performs a line feed as follows:

   a. A hardware device that can be positioned at a particular character position remains positioned immediately after the last character of the operand last displayed.

   b. A hardware device that cannot be positioned at a particular character only suppresses the vertical line feed, if possible. If the hardware device can print new data over the old data, data is displayed over the line already displayed.

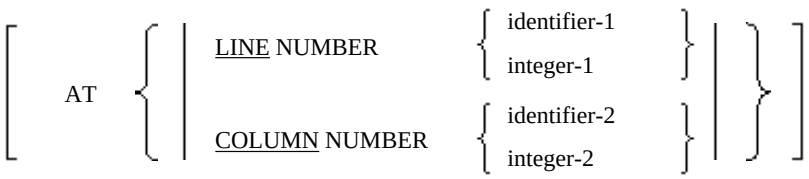
F

6.4.13 DISPLAY Statement (Screen Handling)

The DISPLAY statement displays data on the screen. Note: The screen handling module cannot be used in [Linux], [LinuxIPF], [Linux64] and [.NET].

Format

DISPLAY data-name-1



[END-DISPLAY]

Syntax Rules

1. Data-name-1 must be a screen item defined in the SCREEN SECTION. The screen item must be one of the following:
   a. Elementary screen item which specifies the VALUE clause
   b. Elementary screen item which specifies the FROM or USING phrase in the PICTURE clause.
   c. Group screen item containing an elementary screen item in (a) or (b)
2. Data-name-1 can be qualified.
3. Integer-1 and integer-2 must be unsigned integers.
4. Identifier-1 and identifier-2 must be unsigned integer items.
5. The LINE NUMBER and COLUMN NUMBER phrases can be specified in any order. Either or both phrases can be written.

General Rules

1. The DISPLAY statement displays the following data in the screen field corresponding to the screen item for data-name-1.
   a. If data-name-1 specifies a literal item, the value specified in the VALUE clause is displayed.
   b. If data-name-1 specifies an output or update item, the value of the data item in the FROM or USING phrase of the PICTURE clause is moved to the screen item. Then, the data is displayed. This is done according to the rules of the movement of data.
   c. If data-name-1 specifies a group screen item, all the subordinate screen items belonging to the screen item are displayed.
2. The LINE NUMBER phrase specifies the screen line number corresponding to the screen item for data-name-1. The first line of the physical screen is assumed to be 1.
3. The COLUMN NUMBER phrase specifies the screen column number corresponding to the screen item for data-name-1. The first column of the physical screen is assumed to be 1.
4. If the AT phrase is omitted, data is displayed starting at the first line and column of the screen.

F

6.4.14 DISPLAY Statement (Command Line Arguments and Environmental Variables)

The DISPLAY statement positions a command line argument and an environmental variable, and stores a value in the environmental variable.

Format

```
DISPLAY { identifier-1 } ... UPON mnemonic-name-1
        { literal-1    }
        [ON EXCEPTION imperative-statement-1]
        [NOT ON EXCEPTION imperative-statement-2]
        [END-DISPLAY]
```

Syntax Rules

1. Mnemonic-name-1 must be associated with the following function-name in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION:
   - ARGUMENT-NUMBER

- ENVIRONMENT-NAME
- ENVIRONMENT-VALUE

2. Identifier-1 and literal-1 must satisfy the following conditions:
   a. If mnemonic-name corresponds to the function-name ARGUMENT-NUMBER, identifier-1 must be an unsigned integer item. Also, literal-1 must be an unsigned integer.
   b. If mnemonic-name corresponds to the function-name ENVIRONMENT-NAME or ENVIRONMENT-VALUE, identifier-1 must be a group item not containing a variable occurrence data item, or an alphanumeric data item. Also, literal-1 must be a nonnumeric literal.
3. The ON EXCEPTION and NOT ON EXCEPTION phrases can be specified only if mnemonic-name corresponds to the function-name ENVIRONMENT-VALUE.

General Rules

Rules Applied if Mnemonic-name-1 is Associated with the Function-name ARGUMENT-NUMBER

Specify the position of a command line argument as identifier-1 or literal-1. The values of identifier-1 and literal-1 must be in the range of 0 to 99. Execution of the DISPLAY statement positions the argument position indicator as specified as identifier-1 or literal-1.

Rules Applied if Mnemonic-name-1 is Associated with the Function-name ENVIRONMENT-NAME

Specify the name of an environmental variable as identifier-1 or literal-1. Execution of the DISPLAY statement enables an input-output operation on the environmental variable having the name specified as identifier-1 or literal-1.

Rules Applied if Mnemonic-name-1 is Associated with the Function-name ENVIRONMENT-VALUE

1. Specify the value to be stored in an environmental variable as identifier-1 or literal-1.
2. Before execution of this DISPLAY statement, a DISPLAY statement where mnemonic-name corresponds to the function-name ENVIRONMENT-NAME must be executed. Execution of the DISPLAY statement stores the value specified as identifier-1 or literal-1 in the environmental variable positioned by the preceding DISPLAY statement.
3. If the ON EXCEPTION phrase is specified, imperative-statement-1 is executed under either of the following conditions:
   a. Before execution of the DISPLAY statement, the DISPLAY statement for positioning the environmental variable is not executed.
   b. A sufficient field for storing a value in the environmental variable cannot be acquired.
4. If the NOT ON EXCEPTION phrase is specified, and the exception condition in (2) does not occur, imperative-statement-2 is executed.

6.4.15 DIVIDE Statement (Nucleus)

The DIVIDE statement performs division and obtains the quotient and the remainder.

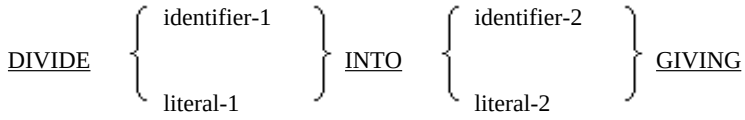
Format 1

To obtain the quotient, and replace the dividend with it:

```
DIVIDE { identifier-1 } INTO {identifier-2 [ROUNDED]} ...
       { literal-1   }
       [ON SIZE ERROR imperative-statement-1]
       [NOT ON SIZE ERROR imperative-statement-2]
       [END-DIVIDE]
```

Format 2

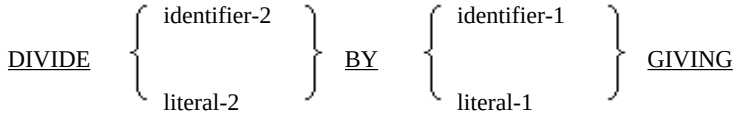
To obtain the quotient, and store it in a data item other than the one used for the dividend:



{identifier-3 [ROUNDED]} ...
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-DIVIDE]

Format 3

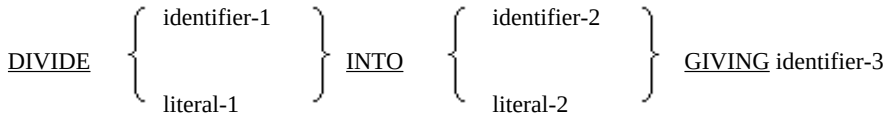
To obtain the quotient, and store it in a data item other than the one used for the dividend:



{identifier-3 [ROUNDED]} ...
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-DIVIDE]

Format 4

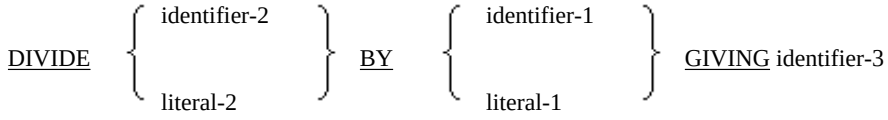
To obtain the quotient and the remainder:



[ROUNDED] REMAINDER identifier-4
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-DIVIDE]

Format 5

To obtain the quotient and the remainder:



[ROUNDED] REMAINDER identifier-4
[ON SIZE ERROR imperative-statement-1]
[NOT ON SIZE ERROR imperative-statement-2]
[END-DIVIDE]

Syntax Rules

1. Identifier-1 and identifier-2 must be numeric data items.
2. Identifier-3 and identifier-4 must be numeric data items or numeric edited data items.
3. Literal-1 and literal-2 must be numeric literals.

General Rules

Rule for Format 1

The format 1 DIVIDE statement divides identifier-1 or literal-1 into identifier-2 and stores the quotient in identifier-2. The division is performed in the order specified by identifier-2.

Rule for Format 2

The format 2 DIVIDE statement divides identifier-1 or literal-1 into identifier-2 or literal-2 by and stores the quotient in identifier-3. The quotient is stored in the order specified by identifier-3.

Rule for Format 3

The format 3 DIVIDE statement divides identifier-2 or literal-2 by identifier-1 or literal-1 stores the quotient in identifier-3. The quotient is stored in the order specified by identifier-3.

Rule for Format 4

The format 4 DIVIDE statement divides the identifier-1 or literal-1 into identifier-2 or literal-2, stores the quotient in identifier-3, and the remainder in identifier-4.

Rule for Format 5

The format 5 DIVIDE statement divides identifer-2 or literal-2 by identifier-1 or literal-1, stores the quotient in identifier-3, and the remainder in identifier-4.

Rules Common to Format 4 and Format 5

1. The subscript in identifier-4, if any, is evaluated immediately before the remainder is stored in identifier-4.
2. The remainder can be obtained by the following expression:

```
Remainder = dividend - quotient x divisor
```

   The dividend is the value of identifier-2 or literal-2. The divisor is the value of identifier-1 or literal-1. One of the following values is used as the quotient to compute the remainder:

   a. If a numeric edited data item is specified as identifier-3, the value is not edited in the format specified in the PICTURE clause.

   b. If the ROUNDED phrase is specified, the value is truncated (before the quotient is rounded off). The following are the same as for identifier-3:

      - The number of the digits of the quotient for computing the remainder

      - Decimal position

      - Whether a sign is included

   c. For other than the above, the value of identifier-3 is used.

3. The precision of identifier-4 observes the rules in (2). A value aligned by a decimal point and truncated is stored in identifier-4 as required.
4. If the ON SIZE ERROR phrase is specified, the execution result are as follows:

   a. If the quotient overflows, the contents of identifier-3 and identifier-4 are not changed.

   b. If the remainder overflows, the content of identifier-4 is not changed. The user must check whether the quotient or the remainder overflows.

Rules Common to All Formats

1. The END-DIVIDE phrase delimits the scope of the DIVIDE statement.

2. For details on the ROUNDED and ON SIZE ERROR phrases, the operation and rules for the movement of data, see the section titled "Common Statement Rules" earlier in this chapter.

F

6.4.16 ENTRY Statement (Inter-program Communication)

The ENTRY statement specifies a secondary entry point for a called program.

Note

The ENTRY statement cannot be used in [.NET].

Format

```
ENTRY literal-1
      [WITH { C        } LINKAGE]
            { PASCAL   }
            { STDCALL  }
      [USING {data-name-1} ... ]
```

[Win16] [Win32] The WITH phrase can be specified.

Syntax Rules

1. Literal-1 must be a nonnumeric literal. Literal-1 must observe the rules for describing program names. For details of the rules, see the topic titled "Program-ID Paragraph" in "Composition of the Identification Division" section of Chapter 3, "Identification Division and End Program Header".
2. Literal-1 must be unique, duplicate entry points and program names are not allowed within a run unit. Outermost program name and secondary entry point names must be unique in a single run unit. But nested programs are allowed to have a same name that separately compiled programs have. Refer to "Scope of a Program-name and Secondary Entry Point Names" in Chapter 2.
3. For details of the rules for the USING phrase, see the topic titled "Program-ID Paragraph" in "Composition of the Identification Division" section of Chapter 3, "Identification Division and End Program Header".
4. For details of the maximum number of operands that can be specified in the USING phrase, see Appendix B, "System Quantitative Restrictions".
5. If the Procedure Division header specifies the RETURNING phrase, the Procedure Division itself should not contain the ENTRY statement.

Rules for the WITH Phrase

The WITH phrase specifies calling conventions. If the WITH phrase is omitted, the COBOL calling convention is applied.

General Rules

1. The ENTRY statement specifies a secondary entry point. The name of the secondary entry point is specified by literal-1. The name of the secondary entry point may be specified as a literal or an identifier in the CALL statement. If so, control is transferred to the secondary entry point when the CALL statement is executed. To receive parameters from the calling program, specify the USING phrase.
2. When the calling program invokes the called program specified in a CALL statement, control is transferred to the statement following the ENTRY statement.
3. Nested programs cannot specify an ENTRY statement.

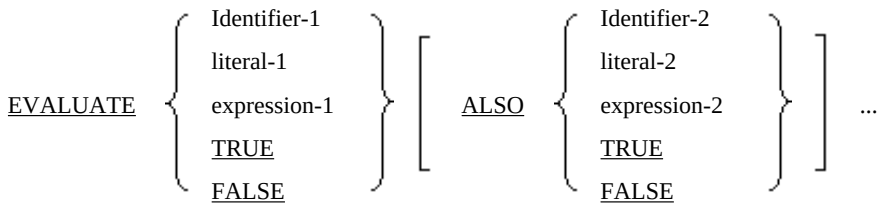
Rules for the WITH Phrase

Within a single separately compiled program, entries in the Procedure Division header of the outermost program must be the same as entries in the WITH phrase of the ENTRY statement.

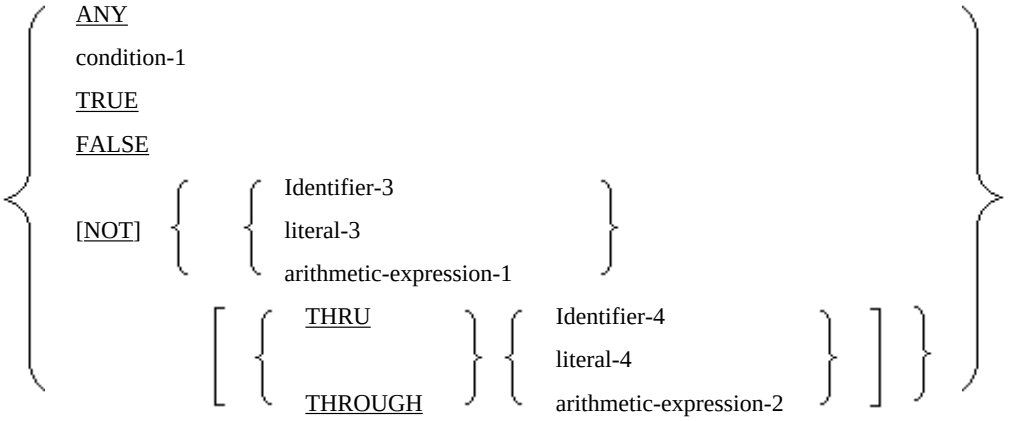
6.4.17 EVALUATE Statement (Nucleus)

The EVALUATE statement evaluates multiple conditions, and executes statements corresponding to the evaluation result.

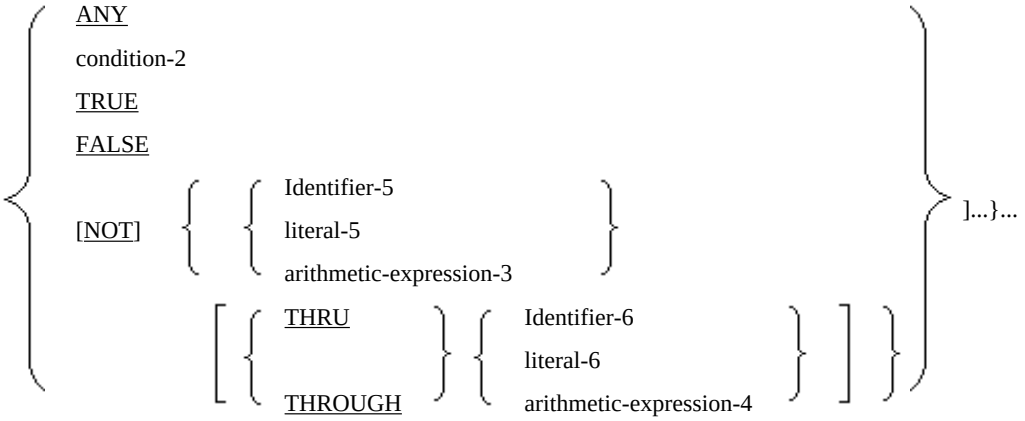
Format



{{WHEN



[ALSO



imperative-expression-1} ...

[WHEN OTHER imperative-expression-2]

[END-EVALUATE]

Syntax Rules

1. The operands (including TRUE and FALSE) before the first WHEN phrase in the EVALUATE statement are called selection subjects.

2. The operands (including TRUE, FALSE, and ANY) in the WHEN phrase in the EVALUATE statement are called selection objects. If the THROUGH phrase is specified in the WHEN phrase, two operands combined by the THROUGH phrase are considered to be one selection object.
3. THRU is synonymous with THROUGH.
4. The two operands combined by the THROUGH phrase must be identifiers, literals, or arithmetic expressions in the same class.
5. The number of the selection objects in one WHEN phrase must equal the number of selection subjects.
6. Each selection object in one WHEN phrase must be associated with a selection subject. For example, the first object must be associated with the first subject. The second object must be associated with the second subject. The rules for associating selection objects with selection subjects are as follows:
    a. When an identifier, literal, or arithmetic expression is specified as a selection object, an operand that can be compared with the object must be written as the corresponding selection subject.
    b. When condition-1, and condition-2, TRUE, or FALSE is specified as a selection object, a conditional expression, TRUE, or FALSE must be written as the corresponding selection subject.
    c. If ANY is specified as a selection object, any corresponding selection subject can be used.
7. Boolean data items or literals cannot be written as the two operands combined by the THROUGH phrase.
8. Boolean expressions cannot be written as expression-1 and expression-2.

General Rules

1. The EVALUATE statement evaluates the values of the individual selection subjects and those of the selection objects. It then selects the WHEN phrase that satisfies the condition, and executes the imperative statement for that WHEN phrase.
2. The following values are assigned to the individual selection subjects and the selection objects:
    a. If identifier-1 or identifier-2 is the selection subject, the value and class of the identifier are assigned to the subject. Similarly, identifier-3 or identifier-5 without the NOT and THROUGH phrases may be a selection object. If so, the value and class of the identifier are assigned to the object.
    b. If literal-1 or literal-2 is the selection subject, the value and class of the literal are assigned to the subject. Similarly, literal-3 or literal-5 without the NOT and THROUGH phrases may be a selection object. If so, the value and class of the literal are assigned to the object. However, if the figurative constant ZERO is specified as literal-3 or literal-5, the class of the selection subject is assigned to that of the corresponding selection object.
    c. An arithmetic expression may be written as expression-1 or expression-2 for a selection subject. If so, the numeric value obtained according to the rules for evaluating an arithmetic expression is assigned to the subject. Similarly, arithmetic-expression-1 or arithmetic-expression-3 without the NOT and THROUGH phrases may be a selection object. If so, the numeric value obtained according to the rules for evaluating an arithmetic expression is assigned to the object.
    d. A conditional expression may be written as expression-1 or expression-2 for a selection subject. If so, the truth value obtained according to the rules for evaluating a conditional expression is assigned to the subject. Similarly, condition-1 or condition-2 may be a selection object. If so, the truth value obtained according to the rules for evaluating a conditional expression is assigned to the object.
    e. If TRUE or FALSE is a selection subject, the truth value is assigned to the subject. If TRUE is specified, true is assigned. If FALSE is specified, false is assigned. Similarly, if TRUE or FALSE is a selection object, the truth value is assigned to the object.
    f. If ANY is a selection object, the selection object is not evaluated.
    g. If the THROUGH phrase is specified, and the NOT phrase is omitted in a selection object, of all the values of the corresponding selection subject, those that satisfy the following conditions are assigned to the selection object:
        - Greater than or equal to the operand immediately before the THROUGH phrase

- Less than or equal to the operand immediately after the THROUGH phrase

h. If the NOT phrase is specified and the THROUGH phrase is omitted in a selection object, the value not equal to the value assigned when the NOT phrase is omitted is assigned to the object. Similarly, the NOT and THROUGH phrases may be specified a selection object. If so, the values not included within the scope of the values assigned when the NOT phrase is omitted are assigned to the object. The values assigned when the NOT phrase is omitted are those explained in items (a) to (h) above.

3. The value of each selection subject and that of the corresponding selection object assigned according to the rules in (2) are compared. Then, the WHEN phrase that satisfies the condition is determined. This comparison is done as follows:

   a. The first selection object in one WHEN phrase is compared with the first selection subject, the second selection object, the second selection subject, and so on. The comparison rules are as follows:

      - The values of selection objects and those of selection subjects may be assigned according to the rules in (a), (b), (c), (g), or (h) of (2). If so, the value of each selection object is compared with that of the corresponding selection subject. If the value assigned to the selection object is the same as that assigned to the corresponding selection subject, the comparison result is true.

      - The value of a selection object may be assigned according to the rules in (d) or (e) of (2). If so, the truth value of the object and that of the corresponding selection subject are compared. If the truth values are identical, the comparison result is true.

      - If ANY is written as a selection object, the comparison result is always true regardless of the value of the corresponding selection subject.

   b. In one WHEN phrase, the results of comparing all selection objects with the corresponding selection subjects in the method (a) may be true. If so, this WHEN phrase is selected as the WHEN phrase that satisfies the condition.

   c. The above steps (a) and (b) are repeated in the order in which the WHEN phrases are written. This is done until the WHEN phrase that satisfies the condition is selected or all the WHEN phrases are processed.

4. After the comparison in (3) is completed, execution of the EVALUATE statement continues as follows:

   a. If the WHEN phrase that satisfies the condition is selected, the first imperative-statement-1 following the selected WHEN phrase is executed. After execution of imperative-statement-1, execution of the EVALUATE statement terminates.

   b. If no WHEN phrase is selected, and the WHEN OTHER phrase was specified, imperative-statement-2 is executed. After execution of imperative-statement-2, execution of the EVALUATE statement terminates.

   c. If no WHEN phrase is selected, and the WHEN OTHER phrase is omitted, execution of the EVALUATE statement terminates.

5. The END-EVALUATE phrase delimits the scope of the EVALUATE statement.

6. For details of conditional and arithmetic expressions, and the comparison rules, see the section titled "Common Statement Rules" earlier in this chapter.

Example

Example of the EVALUATE Statement (1/2)

[DATA DIVISION]

```
01  MAIL-RECORD.
        02 KIND             PIC X.
        02 WEIGHT           PIC 9(4).
        02 CHARGE           PIC 9(5).
        02 FLAG             PIC X(5).
```

[PROCEDURE DIVISION]

```
       EVALUATE KIND ALSO WEIGHT
           WHEN "1" ALSO 0 THRU 25
                  MOVE 60 TO CHARGE
           WHEN "1" ALSO 26 THRU 50
                  MOVE 70 TO CHARGE
           WHEN "2" ALSO ANY
                  MOVE 40 TO CHARGE
           WHEN OTHER
                  MOVE "ERROR" TO FLAG
      END-EVALUATE
```

Using the example shown above, the EVALUATE statement is executed based on the values of KIND and WEIGHT as shown in the following table:

Condition		Statement to be Executed
Value of KIND	Value of WEIGHT	
"1"	0 TO 25	MOVE 60 TO CHARGE
"1"	26 TO 50	MOVE 70 TO CHARGE
"2"	Any value	MOVE 40 TO CHARGE
Other than the above		MOVE "ERROR" TO FLAG

Example of the EVALUATE Statement (2/2)

[DATA DIVISION]

```
01   EMPLOYMENT-RECORD.
        02 SEX            PIC X.
        02 AGE            PIC 9(2).
        02 MARKS          PIC 9(3).
        02 RESULT         PIC X(4).
```

[PROCEDURE DIVISION]

```
       EVALUATE  SEX ALSO AGE ALSO TRUE
           WHEN "F" ALSO 18 THRU 22 ALSO
                  MARKS > 74
           WHEN "F" ALSO 23 THRU 29 ALSO
                  MARKS > 84
           WHEN "M" ALSO 18 THRU 39 ALSO
                  MARKS > 79
                  MOVE "PASS" TO RESULT
                  WHEN OTHER MOVE "FAIL " TO RESULT
      END-EVALUATE
```

Using the example above, the EVALUATE statement is executed based on the values of SEX, AGE, and MARKS as shown in the following table:

Condition			Statement to be executed
Value of SEX	Value of AGE	Value of MARKS	
"F"	18 TO 22	> 74	MOVE "PASS" TO RESULT
"F"	23 TO 29	> 84	
"M"	18 TO 39	> 79	
Other than the above			MOVE "FAIL" TO RESULT

6.4.18 EXIT Statement (Nucleus)

The EXIT statement specifies the exit common to a series of procedures. For using the EXIT statement in [.NET], refer to "EXIT" in "Chapter 12 Microsoft .NET Support".

Format 1

EXIT

F

Format 2

EXIT { PARAGRAPH | SECTION }

Syntax Rules

Rule for Format 1

An EXIT statement must compose a sentence (i.e. be terminated with a period), and this must be the only sentence coded in its paragraph.

Rule for Format 2

An EXIT statement with the SECTION specification must be contained in a section.

General Rules

Rule for Format 1

An EXIT statement is coded only if the user wants to assign a procedure name to a certain point in a program. The EXIT statement does not have any effect on compilation or execution of the program other than as described above.

Rules for Format 2

1. When an EXIT statement with the PARAGRAPH specification is executed, an implicit CONTINUE statement is inserted immediately after the last statement in the paragraph containing this EXIT statement, and control jumps to this CONTINUE statement.
2. When an EXIT statement with the SECTION specification is executed, an implicit CONTINUE statement is inserted immediately after the last statement in the last paragraph in the section containing this EXIT statement, and control jumps to this CONTINUE statement.

F

6.4.19 EXIT PERFORM Statement (Nucleus)

Specifies the exit of an in-line PERFORM statement.

Format

EXIT [TO TEST OF] PERFORM

General Rules

1. The EXIT PERFORM statement specifies the exit of an in-line PERFORM statement. The EXIT PERFORM statement is allowed only within an in-line PERFORM statement.
2. The EXIT PERFORM statement is associated with the innermost PERFORM statement in which it is included.
3. Executing an EXIT PERFORM statement without the TEST phrase specified passes control to the end of the corresponding PERFORM statement.
4. An EXIT PERFORM statement with the TEST phrase specified is allowed only in a PERFORM statement with a termination condition specified (one using format 2, 3, or 4). Executing an EXIT PERFORM statement with the TEST

phrase specified passes control to the inspection process of the corresponding in-line PERFORM statement. The inspection process is either of the following:

a. A process in a format 2 PERFORM statement that inspects the occurrence count specified before the TIMES phrase, or

b. A process in a format 3 or 4 PERFORM statement that inspects the condition specified after the UNTIL phrase.

6.4.20 EXIT PROGRAM Statement (Inter-program Communication)

Specifies the logical end of a called program.

Format

EXIT PROGRAM

Syntax Rules

1. When an EXIT PROGRAM statement is written in a list of imperative statements in a sentence, it must appear at the end of that list.
2. The EXIT PROGRAM is not allowed in the declarative procedure of a USE statement with the GLOBAL phrase specified.

General Rules

1. The EXIT PROGRAM statement specifies the logical end of a called program. If the program executing the EXIT PROGRAM statement is not under the control of a calling program, control is passed to the next executable statement after the EXIT PROGRAM statement. Otherwise, control is passed to the next executable statement in the calling program after the CALL statement.
2. After the EXIT PROGRAM statement is executed, data and files shared by the calling and called program are left in the same state.
3. After the execution of the EXIT PROGRAM statement, the status of the called program (the program that has executed the EXIT PROGRAM statement) depends on whether it has the INITIAL attribute:
   a. If it does not have the INITIAL attribute, it remains in the same state as in effect before it had been called, except that the scope of all PERFORM statements have been determined.
   b. If it has the INITIAL attribute, it enters the same status as it would upon execution of a CANCEL statement.

6.4.21 GENERATE Statement (Report writer)

Outputs a report according to a report description entry.

Note: The report writer module cannot be used in [Winx64], [LinuxIPF] and [.NET].

Format

GENERATE { data-name-1 / report-name-1 }

Syntax Rules

1. Data-name-1 must name a detail report group. Data-name-1 may be qualified by a report name.
2. Report-name-1 can only be specified if the report description entry meets all of the following requirements:
   a. The report description entry contains a CONTROL clause.
   b. The report description entry does not contain more than one detail report group.

c. The report description entry contains one or more report body groups.

General Rules

1. The report writer control system performs summary report processing on GENERATE statements with report-name-1 specified. If all the GENERATE statements for a report specify report-name-1, the report that is subsequently generated and presented is called a summary report. A summary report is a report in which a detail report group is not output.
2. The report writer control system performs detail processing on GENERATE statements with data-name-1 specified, including processing specific to the detail report group specified. Normally, executing GENERATE statements with data-name-1 specified causes the report writer control system to present the specified detail report group.
3. The report writer control system saves the value of control data items when it executes the first GENERATE statement for the report. The system uses these values to detect control breaks when subsequent GENERATE statements for the report are executed. When the system detects a control break, it saves the new values of the control data items, using these values to detect further control breaks. This process is repeated until the report is closed by execution of the TERMINATE statement.
4. When a need arises to perform a form feed to present the report body group during report presentation, the report writer control system automatically executes processing as specified by a page heading report group and a page footing group if they are defined.
5. The report writer control system executes processing as specified by the page heading report group and page footing group as explained in (4), processing the following report groups in the order of their appearance in the report description entry when it executes the first GENERATE statement for the report:
    a. Report heading report group
    b. Page heading report group
    c. Control heading groups at all levels, from high to low
    d. If a GENERATE statement with data-name-1 specified is executed, the detail report group specified is processed; if a GENERATE statement with report-name-1 specified is executed, part of the detail report processing
6. The report writer control system executes processing as specified by the page heading report group and page footing group as explained in (4), performing the following operations when it executes a second or subsequent GENERATE statement for the report:
    a. Detection of control breaks. The rules of relational conditions apply to determining the equality of control data items. When a control break is detected, the report writer control system:
        - Makes the value of the control data item used to detect the control break accessible for reference to the USE procedure associated with the control footing group and the SOURCE clause in the control footing group.
        - Processes the control footing groups in the ascending order of levels. All the control footing groups at any level lower than the level in effect when the control break occurred are processed.
        - Processes the control heading groups in the descending order of levels. All the control heading groups at any level lower than the level in effect when the control break occurred are processed.
    b. If a GENERATE statement with data-name-1 specified is executed, the detail report group specified is processed; if a GENERATE statement with report-name-1 specified is executed, part of the detail report processing.
7. GENERATE statements for a report can only be executed after the execution of an INITIATE statement and before the execution of a TERMINATE statement.
8. For the report presentation rules, see the topic titled "Type Clause" in "Report Group Description Entry" section of Chapter 5 and the section titled "Report Group Presentation Rules" of Chapter 5, the "Data Division".

6.4.22 GO TO Statement (Nucleus)

Passes control from one point in the PROCEDURE DIVISION to another. A format 1 GO TO statement with procedure-name-1 omitted is an obsolete element related to the ALTER statement.

Format 1

Passes control to a specific point.

GO TO [procedure-name-1]

Format 2

Passes control according to the value of a data item.

GO TO {procedure-name-1} ... DEPENDING ON identifier-1

Syntax Rules

Rules for Format 1

1. When the destination of the GO TO statement is altered with an ALTER statement, the paragraph specified in the ALTER statement must be composed of a single sentence consisting of a format 1 GO TO statement .
2. A GO TO statement without procedure-name-1 specified must be written in a single sentence. This statement is only allowed in a single statement paragraph.
3. When a GO TO statement is written in a statement consisting of a number of imperative statements, it must be the last imperative statement.

Rule for Format 2

Identifier-1 must be an integer item.

General Rules

Rules for Format 1

1. A format 1 GO TO statement passes control to procedure-name-1.
2. When procedure-name-1 is omitted, the execution of the GO TO statement must be preceded with the execution of an ALTER statement that references the GO TO statement.

Rule for Format 2

Specify the destinations to which control passes when the value of identifier-1 is 1, 2, 3, ..., *n*, in the procedure-name-1 list in sequence from left to right. A format 2 GO TO statement passes control the to first procedure-name-1 if the value of identifier-1 is 1, to the second procedure-name-1 if the value of identifier-1 is 2, and so on. The GO TO statement passes control to the next statement if the value of identifier-1 is not 1, 2, 3, ..., *n*.

6.4.23 IF Statement (Nucleus)

Evaluates a condition and executes the statement corresponding to the result of the evaluation.

Format

IF condition-1

THEN { {statement-1}... | NEXT SENTENCE }

```
{ ELSE {statement-2}... [END-IF]
  ELSE NEXT SENTENCE
  END-IF                        }
```

Syntax Rules

1. The ELSE NEXT SENTENCE phrase can be omitted when it is immediately followed by a separator period.
2. The NEXT SENTENCE phrase and the END-IF phrase cannot both be specified.

General Rules

1. The IF statement evaluates condition-1 and executes the operation specified by the THEN phrase if it is true or the operation specified by the ELSE phrase if it is false.
2. The IF statement is terminated by either:
   a. An END-IF phrase appearing at the same level
   b. A separator period
   c. The ELSE phrase in the outermost IF statement if multiple IF statements are nested.

   For the scope of the IF statement, see the topic "Scope of Statements" of the "Composition of the Procedure Division" section earlier in this chapter.
3. If the result of the evaluation of condition-1 is true, control is passed as follows:
   a. If statement-1 is specified, the statement(s) specified in the statement-1 list is executed. Control then passes to the end of the IF statement after statement-1 is executed. However, statement-1 may transfer control to another location.
   b. If NEXT SENTENCE is specified, control passes to the sentence following the IF statement.
4. If the result of the evaluation of condition-1 is false, control will pass to the ELSE phrase. Control is passed as follows:
   a. If statement-2 is specified, the statement(s) specified in the statement-2 list is executed. Control passes to the end of the IF statement when the execution of statement-2 is completed. However, statement-2 may transfer control to another location.
   b. When the ELSE phrase is omitted, control passes to the end of the IF statement.
   c. When NEXT SENTENCE is written, control passes to the sentence following the IF statement.
5. Where IF statements are nested, one IF statement nested in another is assumed to form a set of IF, ELSE, and END-IF from left to right.
6. The END-IF statement delimits the scope of an IF statement.
7. For more details on conditional expressions and comparison rules, see the section titled "Common Statement Rules" earlier in this chapter.

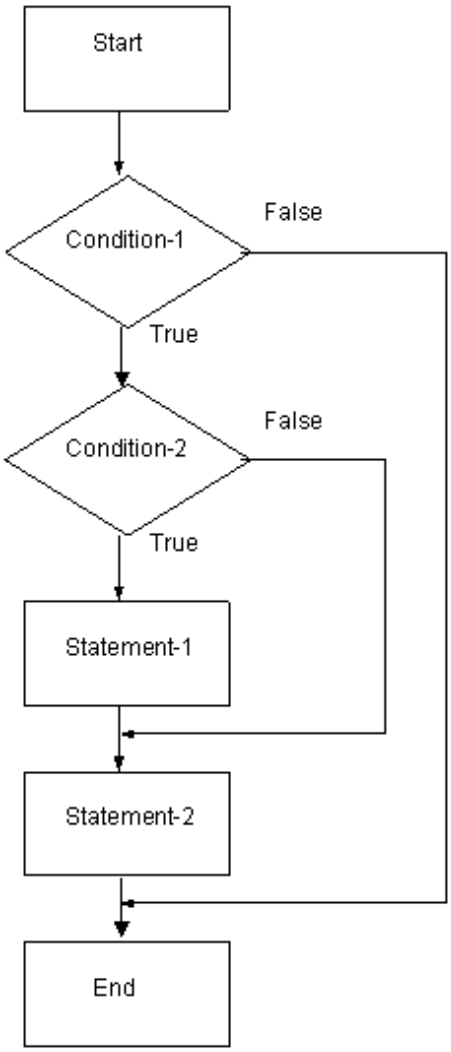
Example

Example of Nested IF Statements

An example of nested IF statements is shown below.

```
IF condition-1 THEN
   IF condition-2 THEN
     *> statement-1
   END-IF
   *> statement-2
END-IF
```

Logical flow of processing by nested IF statements



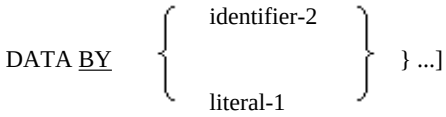
6.4.24 INITIALIZE Statement (Nucleus)

Initializes a data item.

For details about how to use the examples, refer to "INITIALIZE Statement" in the "Syntax Samples".

Format

```
INITIALIZE {identifier-1} ...  [WITH FILLER]
[REPLACING{ ALPHABETIC
            ALPHANUMERIC
            NUMERIC
            ALPHANUMERIC-EDITED
            NUMERIC-EDITED
            NATIONAL
            NATIONAL-EDITED
            BOOLEAN
          }]
```



F The WITH FILLER phrase is specific to [Win32], [Winx64], [Linux64] and [.NET].

Syntax Rules

1. The combination of the category of identifier-2 or literal-1 and the category specified in the REPLACING phrase must conform to the rules of the transcription and movement of data in which the former is a sending category and the latter is a receiving category.
2. A REPLACING phrase of the same category may not be duplicated in the REPLACING phrase list.
3. An OCCURS clause with the DEPENDING phrase may not be specified in identifier-1 or in the data description entry of a data item subordinate to identifier-1.
4. Identifier-1 and identifier-2 may not reference an index data item.
5. Identifier-1 may not reference a data item with the RENAME clause.

General Rules

1. The INITIALIZE statement initializes the identifier-1data item. Leave the REPLACING phrase unspecified to initialize a data item of any category with a predefined value. Specify the REPLACING phrase to initialize an elementary item of a particular category with a particular value. The category of the data to be initialized is specified after the REPLACING keyword, and the initial value of the data items of that category is specified identifier-1 or literal-1.
2. The word that follows REPLACING specifies the category to be the object of initialization.
   - ALPHABET represents alphabetic data items
   - ALPHANUMERIC represents alphanumeric data items
   - NUMERIC represents numeric data items
   - ALPHANUMERIC-EDITED represents alphanumeric edited data items
   - NUMERIC-EDITED represents numeric data items
   - F NATIONAL represents national data items
   - NATIONAL-EDITED represents national edited data items
   - BOOLEAN represents Boolean data items
3. If identifier-1 is an index or pointer data item(*), the execution of the INITIALIZE statement does not alter the contents of identifier-1.
4. If identifier-1 is a group item, the following subordinate elementary items are not initialized:
   - index data item
   - F point data item(*)
   - FILLER item

     F However, when the WITH FILLER phrase is specified, the FILLER item is initialized.
   - A data item specifying the REDEFINES clause or any data item subordinate to such a data item.

5. If identifier-1 or a subordinate elementary item of identifier-1 is the following item, it is initialized with the value defined when the REPLACING phrase was omitted. This rule is specific to [Win32], [Winx64] and [Linux64].

   - When the category specified in the REPLACING phrase is alphanumeric or alphanumeric edited:

     An alphanumeric or alphanumeric edited data item whose encoding form is not the same as Identifier-2's.

   - When the category specified in the REPLACING phrase is national or national edited:

     A national or national edited data item whose encoding form is not the same as Identifier-2's.

6. When the REPLACING phrase is specified, data item initialization takes place as follows:

   a. If identifier-1 is an elementary item, it is initialized with the value of identifier-2 or literal-1 only if that elementary item belongs to the category specified in the REPLACING phrase, except in case (3) and (5) above.

   b. If identifier-1 is a group item, subordinate elementary items belonging to the category specified in the REPLACING phrase are initialized with the value of identifier-2 or literal-1. If the group item includes a table, the elementary items that form the table elements are also subject to initialization, except in case (4) and (5) above.

   c. Initialization takes place as though a MOVE statement is executed where the operand specified with BY phrase is the sending item and the elementary item that is subject to initialization is the receiving item.

7. When the REPLACING phrase is omitted, data item initialization takes place as follows:

   a. If identifier-1 is an elementary item, it is initialized with the value described in (c) below.

   b. If identifier-1 is a group item, the subordinate elementary items are initialized with the value described in (c) below. If the group item includes a table, the elementary items that form the table elements are also subject to initialization, except in case (4) above.

   c. Initialization takes place as though a MOVE statement is executed in which a literal having any of the values described below is the sending item and the elementary item that is subject to initialization is the receiving item.

      - If the elementary item that is subject to initialization has the category alphabetic, alphanumeric or alphanumeric edited, the value of the sending item is an alphanumeric space.

F   - If the elementary item that is subject to initialization has the category national, or national edited, the value of the sending item is a national space.

      - If the elementary item that is subject to initialization has the category numeric or numeric edited, the value of the sending item is a numeric zero.

F   - If the elementary item that is subject to initialization has the category Boolean, the value of the sending item is a Boolean zero.

8. If identifier-1 is a group item, the subordinate elementary items that are subject to initialization are initialized in the order they are defined within the group.

9. If identifier-1 and identifier-2 occupy the same storage area, the result of the execution of the INITIALIZATION statement is unpredictable.

F * : Extended functions or functions specific to NetCOBOL.

- pointer data item

6.4.25 INITIATE Statement (Report writer)

Initiates a report writing process.

Note

The report writer module cannot be used in [Winx64], [LinuxIPF] and [.NET].

Format

INITIATE {report-name-1} ...

Syntax Rules

Report-name-1 must be defined in a report description entry in the REPORT SECTION of the DATA DIVISION.

General Rules

1. The INITIATE statement executes the following initialization processes for the named report.
   a. Resetting the sum counter to 0
   b. Resetting the line counter to 0.
   c. Setting the page counter to 1.
2. The INITIATE statement does not open the named report file. To open such files, an OPEN statement with the OUTPUT or EXTEND phrase specified must be executed before execution of the INITIATE statement.
3. After executing the INITIATE statement for report-name-1, the INITIATE statement cannot again be executed without executing the TERMINATE statement.
4. If two or more report names are specified in a single INITIATE statement, its execution would produce the same result as the execution of separate INITIATE statements in which the report names are written in the order of their appearance in the single INITIATE statement.

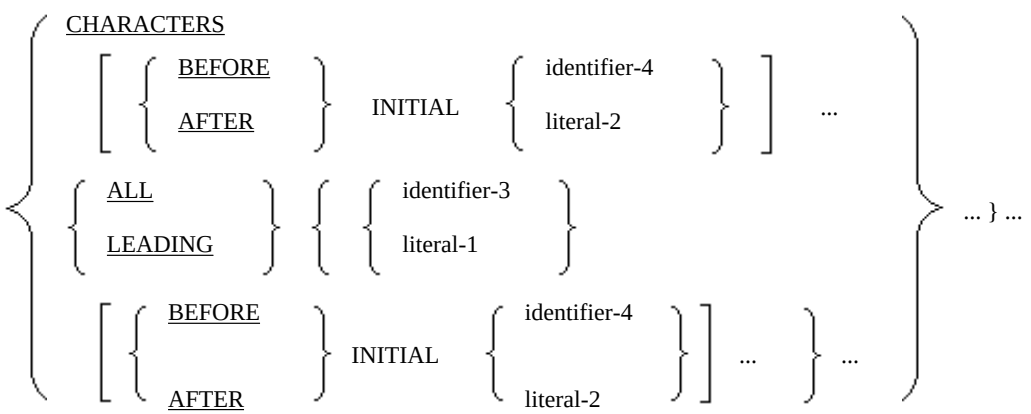
6.4.26 INSPECT Statement

Counts the number of occurrences of a character string (format 1), replacing them (format 2), counts and replaces character strings (format 3), or converts character strings (format 4).

For details about how to use the examples, refer to "INSPECT Statement" in the "Syntax Samples".

Format 1

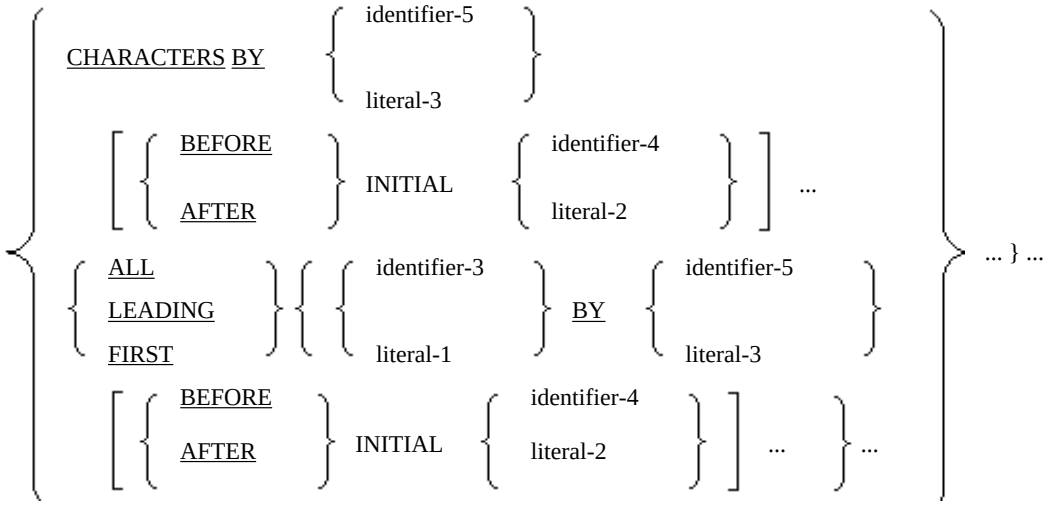
Count the occurrences of a character string.



Format 2

Replaces a character string.

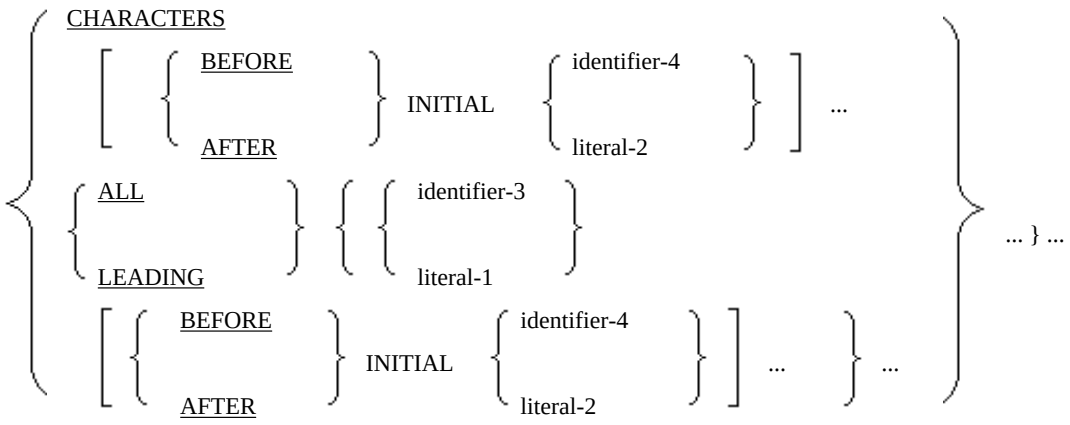
INSPECT identifier-1 REPLACING



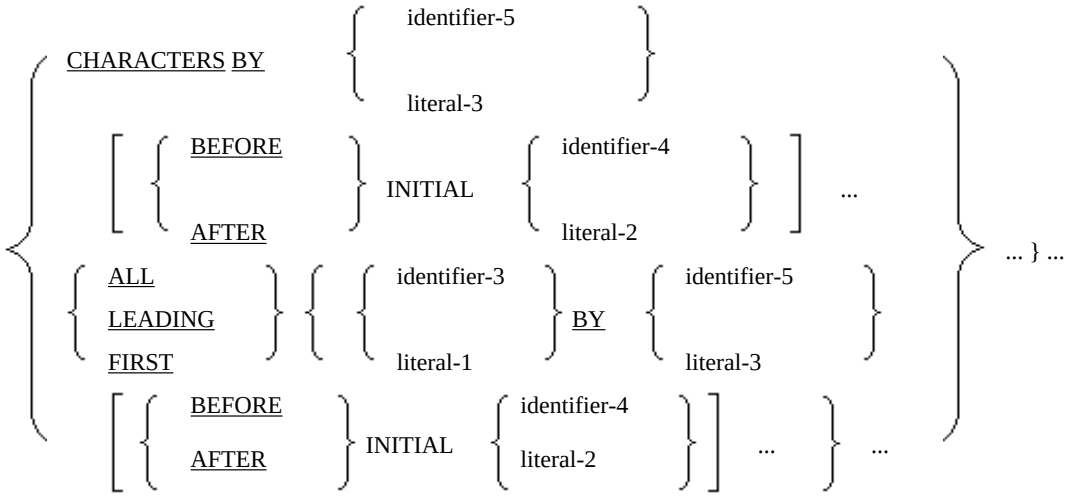
Format 3

Count the occurrences of a character string and replace the string.

INSPECT identifier-1 TALLYING {identifier-2 FOR



REPLACING



Format 4

Specifies replacing character strings in a group.

```
INSPECT identifier-1 CONVERTING
{ identifier-6 } TO { identifier-7 }
{ literal-4   }    { literal-5   }
[ { BEFORE } INITIAL { identifier-4 } ] ...
{ AFTER  }           { literal-2   }
```

Syntax Rules

Rules for All Formats

1. Identifier-1 must be a group item or an elementary item whose use is presentation.
2. Identifier-2 must be a numeric data item.
3. Identifier-3 to identifier-7 must each be a group item or an elementary item whose use is presentation.
4. Literal-1 to literal-5 must each be a nonnumeric literal or national literal, or a figurative constant that does not begin with ALL keyword. If a figurative constant is specified in literal-1, literal-2, or literal-4, its length is assumed one character.
5. Only one BEFORE phrase and only one AFTER phrase may follow each ALL, LEADING, FIRST, CHARACTERS, OR CONVERTING phrase.
6. Combinations of identifier-3 to identifier-7 and literal-1 to literal-5 must conform to the following rules:
   a. If identifier-1 is a national data item or national edited data item, identifier-3 to identifier-7 must each be a national data item or national edited data item. Literal-1 to literal-5 must each be a national literal.
   b. If identifier-1 is not a national data item or national edited data item, identifier-3 to identifier-7 must not be a national data item or national edited data item. Literal-1 to literal-5 must each not be a national literal.
   c. The encoding form of identifier-3 to identifier-7 and identifier-1 must be the same. This rule is specific to [Win32], [Winx64] and [Linux64].

Rules Common to Format 2 and Format 3

1. Literal-3 and identifier-5 in the CHARACTERS phrase must be one character long.
2. Literal-3 and identifier-5 in the ALL, LEADING, or FIRST phrase must be equal in their character length to literal-1 or identifier-3. If a figurative constant is specified in literal-3, it is assumed equal in its character length to literal-1 or identifier-3.

Rules for Format 4

1. Literal-5 and identifier-7 must be equal in their character length to literal-4 or identifier-6. If a figurative constant is specified in literal-5, it is assumed equal in its character length to literal-4 or identifier-6.
2. The same character may not appear more than once in the character string of literal-4 or identifier-6.
3. For the maximum permissible character lengths of identifier-6 and identifier-7, see Appendix B, "System Quantitative Restrictions".

General Rules

Rules for Format 1

1. A format 1 INSPECT statement checks to see if the character string (inspected character string) specified in the FOR phrase is included in identifier-1 and counts its occurrences. The scope of character string inspection in identifier-1 is specified in the BEFORE or AFTER phrase. The CHARACTERS, ALL, or LEADING phrase specifies the way the occurrences of the inspected character string are tallied. The inspected character string is specified with identifier-3 or literal-1. If the CHARACTERS phrase is specified, all the occurrences of the character string falling in the scope of character string inspection in identifier-1 are tallied.

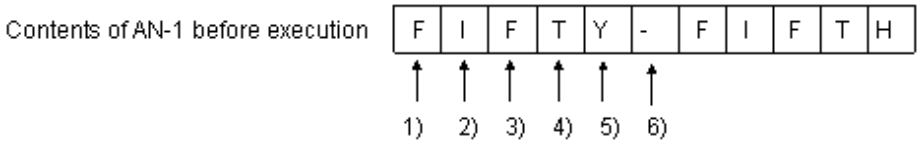
2. If two or more inspected character strings are specified in the FOR phrase, they are inspected in the order of their entry.
3. The character string specified in identifier-1 is inspected once from left to right in sequence.
4. If two or more inspected character strings appear in a single ALL or LEADING phrase, ALL or LEADING applies to each inspected character string.
5. An initial value must be assigned to identifier-2 prior to execution of the INSPECT statement.
6. If identifier-1, identifier-3 or identifier-4 occupies the same storage area as identifier-2, the validity of the result of execution of the INSPECT statement is undefined.

- Processing Flow of a Format 1 INSPECT Statement

The processing flow discussed below references the following INSPECT statement:

Example of a format 1 INSPECT statement

```
    INSPECT AN-1 TALLYING ED-2 FOR ALL "F"
                BEFORE  INITIAL "-".
```



a. The inspected character string is "F."
b. The character string AN-1 is compared with the inspected character string beginning with the leftmost character, one character at a time. 1) to 6) denote the leftmost character position at each instant of comparison. The comparison terminates at 6) (it stops when the initial period, "." is encountered) where the scope of the character string inspection is exceeded.
c. The value of ED-2 is incremented by 1 during the comparison of the characters at 1) and 3) as they match the inspected character string. If the value of ED-2 before the execution of the INSPECT statement is 0, it is incremented to 2 after the execution of the INSPECT statement. If the value of ED-2 is not zero, the value is incremented by 2.

Rules for Format 2

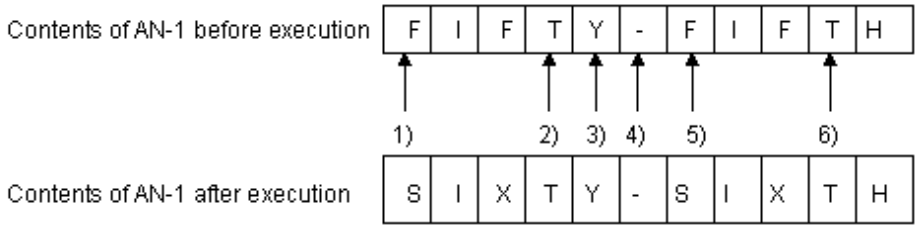
1. The format 2 INSPECT statement checks to see if the character string (inspected character string) specified in the REPLACING phrase is included in identifier-1 and replaces each occurrence of the character string matching the inspected character string with identifier-5 or literal-3. The scope of character string inspection in identifier-1 is specified in the BEFORE or AFTER phrase. The CHARACTERS, ALL, LEADING, or FIRST phrase specifies the way the occurrences of the inspected character string are replaced. The inspected character string is specified with identifier-3 or literal-1. If the CHARACTERS phrase is specified, all occurrences of the character string falling in the scope of character string inspection in identifier-1 are replaced.
2. If two or more inspected character strings are specified in the REPLACING phrase, they are inspected in the order specified.
3. The character string specified in identifier-1 is inspected once from left to right in sequence.
4. If two or more inspected character strings appear in a single ALL, LEADING, or FIRST phrase, ALL, LEADING, or FIRST applies to each inspected character string.
5. If identifier-3, identifier-4, or identifier-5 occupies the same storage area as identifier-1, the validity of the result of execution of the INSPECT statement is undefined.

- Processing Flow of a Format 2 INSPECT Statement

The processing flow of a format 2 INSPECT statement illustrated below references the following INSPECT statement:

Example of a format 2 INSPECT statement

```
INSPECT AN-1 REPLACING ALL "FIF" BY "SIX".
```



a. The inspected character string is "FIF."

b. The character string AN-1 is compared with the inspected character string from the leftmost character, three characters at a time. 1) to 6) denote the leftmost character position at each instant of comparison.

c. "FIF" is replaced with "SIX" during the comparison of the character strings starting at 1) and 5) as they match the inspected character string.

Rules for Format 3

1. The format 3 INSPECT statement is equivalent to a format 1 INSPECT statement with only the TALLYING phrase specified, followed by a format 2 INSPECT statement with only the REPLACING phrase specified. For example, the INSPECT statement in (a) and that in (b) would yield the same result when executed.

   (a)

```
INSPECT AN-1 TALLYING ED-2  FOR  ALL   "F"
                               BEFORE INITIAL "-"
                               REPLACING ALL "FIF" BY  "SIX".
```

   (b)

```
INSPECT AN-1  TALLYING ED-2  FOR ALL  "F"
                               BEFORE INITIAL "-".
INSPECT  AN-1 REPLACING ALL "FIF" BY "SIX".
```

2. The occurrence calculation process is governed by the format 1 rules, and the replacement process is governed by the format 2 rules.

Rules for Format 4

1. A format 4 INSPECT statement in is equivalent to a format 2 INSPECT statement with the REPLACING phrase followed by two ALL phrases and nothing else.

   The following correspondence exists between the operand specified in format 4 and that specified in format 2:

   - Identifier-1 in format 4 : Identifier-1 in format 2.
   - One character at a time as specified in identifier-6 or literal-4 in format 4 : Identifier-3 or literal-1 in format 2 with the ALL phrase specified.
   - One character at a time as specified in identifier-7 or literal-5 in format 4 : Identifier-5 or literal-3 in format 2 with the ALL phrase specified.
   - Identifier-4 or literal-2 in format 4 : Identifier-4 or literal-2 in format 2 with the ALL phrase specified.

   For example, the execution of the INSPECT statement in (a) and that of the INSPECT statement in (b) would yield the same result.

   (a)

```
INSPECT ITEM CONVERTING
       "ABCD" TO "XYZX" AFTER QUOTE BEFORE "@".
```

   (b)

```
INSPECT ITEM REPLACING
        ALL "A" BY "X" AFTER QUOTE BEFORE "@"
        ALL "B" BY "Y" AFTER QUOTE BEFORE "@"
        ALL "C" BY "Z" AFTER QUOTE BEFORE "@"
        ALL "D" BY "X" AFTER QUOTE BEFORE "@".
```

2. If identifier-4, identifier-6 or identifier-7 occupies the same storage area as identifier-1, the validity of the result of execution of the INSPECT statement are undefined.

Rules Common to Format 1 to Format 4

- identifier rules

  All identifiers, except for identifier-2, are assumed to have the contents described below.

  1. If an alphanumeric data item, alphabetic data item, or National data item is specified as an identifier, its contents are assumed to be a character string.
  2. If an alphanumeric edited data item, numeric edited data item, unsigned numeric data item or external Boolean data item is specified as an identifier, its contents are assumed to be a character string redefined by an alphanumeric data item.
  3. If a national edited data item is specified as an identifier, its contents are assumed to be a character string redefined by a National data item.
  4. If a signed numeric edited data item is specified as an identifier, its contents are moved to an unsigned numeric data item of the same length and are assumed to be a character string. The sign of the identifier is saved until the INSPECT statement is completed.
  5. If the encoding form of subordinate elementary items of identifier-1 are not the same, the result of executing the INSPECT statement is undefined.

- Rules governing the evaluation of subscripts and reference modifiers

  If identifier-1 to identifier-7 are subscripted or reference-modified or if a function-identifier is specified for any identifier other than identifier-2, the subscript, reference identifier, and function-identifier are evaluated only once at the start of the execution of the INSPECT statement.

- Scope of character string inspection

  1. If the BEFORE and AFTER phrases are omitted, all the character string specified by identifier-1 are subject to character string inspection.
  2. If the BEFORE phrase is specified, those character strings specified by identifier-1 that intervene between the leftmost character position and the character string just before the first character string that matches identifier-4 or literal-2 are subject to character string inspection. The rightmost character position in the scope of character string inspection is determined before the start of the comparison with the inspected character string. If no characters are found among the character strings specified by identifier-1 that match identifier-4 or literal-2, the BEFORE phrase is assumed to have been omitted.
  3. If the AFTER phrase is specified, those character strings specified by identifier-1 that intervene between the character string just after the first character string that matches identifier-4 or literal-2 and the rightmost character position are subject to character string inspection. The leftmost character position in the scope of character string inspection is determined after the start of the comparison with the inspected character string. If no characters are found among the character strings specified by identifier-1 that match identifier-4 or literal-2, the comparison with the inspected character string is suppressed.

- Procedures for comparison with the inspected character string

  1. A comparison with an inspected character string takes place in the sequence described below. The first term "current leftmost position" below refers to the leftmost character position in the scope of inspection for the character strings specified by identifier-1.

     (a) The character string that begins at the "current leftmost position" is compared with the inspected character string the number of times equal to the number of characters in the inspected character string.

(b) If the character string matches in the comparison described in (a), all its occurrences are tallied and replaced according to the rules for "Counting Occurrences and Replacing Character Strings" explained below. Then, the "current leftmost character position" moves to the character position just to the right of the rightmost character in the character string compared.

(c) When the character string does not match in the comparison described in (a), if two or more inspected character strings are specified, the character string beginning at the "current leftmost position" is compared with the inspected character string in the order of their entry. This comparison is repeated until the character string matches or there are no more inspected character strings. If the character string matches, the same operation as explained in (b) is executed. If there are no more inspected character strings, the "current leftmost character position" moves to the character position just to the right of the rightmost character in the character string compared.

(d) The operations explained in (a) through (c) are repeated until the "current leftmost character position" exceeds the scope of inspection for the character strings specified by identifier-1.

2. If the CHARACTERS phrase is specified, one implicit character is assumed as a inspected character string and all the characters are assumed to match the inspected character string at all times in the inspected character string comparison.

- Counting Occurrences and Replacing Character Strings

1. If the ALL phrase is specified, all the occurrences of the character string falling in the scope of that match the inspected character string are tallied and replaced.
2. If the LEADING phrase is specified, only those occurrences of the character string, beginning at the leftmost position in the scope of character string inspection, that match the inspected character string are tallied and replaced until a character string different from the inspected character string appears.
3. If the FIRST phrase is specified, the occurrences of only the first character string to match the inspected character string in the scope of character string inspection are tallied and replaced.
4. If the CHARACTERS phrase is specified, the occurrences of all the character strings in the scope of character string inspection are tallied and replaced.

Example

INSPECT Statement Example (1)

(a)

```
INSPECT ITEM TALLYING
  COUNT-0 FOR ALL "AB",  ALL "D"
  COUNT-1 FOR ALL "BC"
  COUNT-2 FOR LEADING "EF"
  COUNT-3 FOR LEADING "B"
  COUNT-4 FOR CHARACTERS.
```

(b)

```
INSPECT ITEM REPLACING
  ALL "AB" BY "XY",  "D" BY "X"
  ALL "BC" BY "VW"
  LEADING "EF" BY "TU"
  LEADING "B" BY "S"
  FIRST "G" BY "R"
  FIRST "G" BY "P"
  CHARACTERS BY "Z".
```

The table below summarizes the results of execution of the INSPECT statements (a) and (b).

Initial value of ITEM	Result of execution of (a) (*1)					Result of execution of (b)
	COUNT-0	COUNT-1	COUNT-2	COUNT-3	COUNT-4	Final value of ITEM
EFABDBCGABCFGG	3	1	1	0	5	TUXYXVWRXYZZPZ
BABABC	2	0	0	1	1	SXYXYZ
BBBC	0	1	0	2	0	SSVW

*1 The initial values of COUNT-0 to COUNT-4 are all assumed 0.

INSPECT Statement Example (2)

(a)

```
INSPECT ITEM TALLYING
  COUNT-0 FOR CHARACTERS
  COUNT-1 FOR ALL "A".
```

(b)

```
INSPECT ITEM REPLACING
  CHARACTERS BY "Z"
  ALL "A" BY "X".
```

The following table summarizes the results of execution of the INSPECT statements (a) and (b).

Initial Value of Item	Result of Execution of (a)(*1)		Result of Execution of (b)
	COUNT-0	COUNT-1	Final value of ITEM
BBB	3	0	ZZZ
ABA	3	0	ZZZ

*1 The initial values of COUNT-0 and COUNT-2 are all assumed 0.

INSPECT Statement Example (3)

(a)

```
INSPECT ITEM TALLYING
  COUNT-0 FOR ALL "AB" BEFORE "BC"
  COUNT-1 FOR LEADING "B" AFTER "D"
  COUNT-2 FOR CHARACTERS AFTER "A" BEFORE "C".
```

(b)

```
INSPECT ITEM REPLACING
  ALL "AB" BY "XY" BEFORE "BC"
  LEADING "B" BY "W" AFTER "D"
  FIRST "E" BY "V" AFTER "D"
  CHARACTERS BY "Z" AFTER "A" BEFORE "C".
```

The table below summarizes the results of execution of the INSPECT statements (a) and (b).

Initial Value of ITEM	Result of execution of (a) (*1)			Result of execution of (b)
	COUNT-0	COUNT-1	COUNT-2	Final value of ITEM
BBEABDABABBCABEE	3	0	2	BBEXYZXYXYZCABVE
ADDDDC	0	0	4	AZZZZC

Initial Value of ITEM	Result of execution of (a) (*1)			Result of execution of (b)
	COUNT-0	COUNT-1	COUNT-2	Final value of ITEM
ADDDDA	0	0	5	AZZZZZ
CDDDDC	0	0	0	CDDDDC
BDBBBDB	0	3	0	BDWWWDB

*1 The initial values of COUNT-0 to COUNT-2 are all assumed 0.

INSPECT Statement Example (4)

(a)

```
    INSPECT ITEM TALLYING
      COUNT-0 FOR ALL "AB" AFTER "BA" BEFORE "BC".
```

(b)

```
    INSPECT ITEM REPLACING
      ALL "AB" BY "XY" AFTER "BA" BEFORE "BC".
```

The following table summarizes the results of execution of the INSPECT statements (a) and (b).

Initial value of ITEM	Result of execution of (a) (*1)	Result of execution of (b)
	COUNT-0	Final value of ITEM
ABABABABC	1	ABABXYABC

*1 The initial value of COUNT-0 is assumed 0.

INSPECT Statement Example (5)

```
    INSPECT ITEM CONVERTING
      "ABCD" TO "XYZX" AFTER QUOTE BEFORE "@".
```

The table below summarizes the result of execution of this INSPECT statement.

Initial value of ITEM	Final value of ITEM
AC"AEBDFBCD@AB"D	AC"XEYXFYZX@AB"D

6.4.27 MERGE Statement (Sort-merge)

Merges two or more files together.

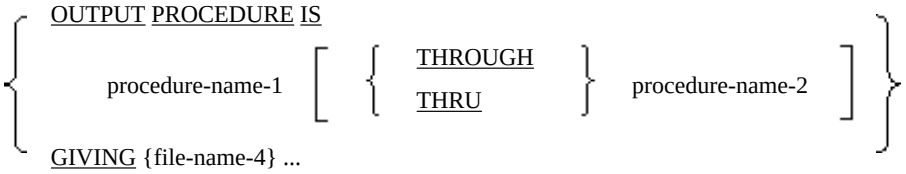
Format

MERGE file-name-1

{ON { ASCENDING / DESCENDING } KEY {data-name-1}...}...

[COLLATING SEQUENCE IS alhabet-name-1]

USING file-name-2 {file-name-3}...



Syntax Rules

1. The MERGE statement may appear anywhere in the PROCEDURE DIVISION, except in the declaratives.
2. File-name-1 must be defined in sort merge file description entry of DATA DIVISION.
3. File-name-2 and file-name-3 must conform to the following rules:
    a. If the file identified by file-name-1 has a variable-length record format, the record length of the files identified by file-name-2 and file-name-3 must be greater than or equal to the minimum record length of the file identified by file-name-1 but must not exceed the maximum record length.
    b. If the file identified by file-name-1 has a fixed-length record format, the record length of the files identified by file-name-2 and file-name-3 must not exceed that of file-name-1.
4. Data-name-1 must conform to the following rules:
    a. Data-name-1 must be a data item defined in the record description entry of file-name-1.
    b. Data-name-1 may be qualified.
    c. A group item having a variable occurrence data item subordinate to it may not be specified in data-name-1.
    d. If two or more record description entries are specified in file-name-1, the data item specified in one record description entry must be specified in the data-name-1 list. The same character position as the data item in data-name-1 in that record description entry is assumed a key in every record in that file.
    e. A data item in which an OCCURS clause is specified or a data item that is subordinate to such a data item may not be specified with data-name-1.
    f. If file-name-1 has a variable-length record format, the data item of data-name-1 must be included in the range of the size of the smallest record in file-name-1.
    g. Data-name-1 may not be a Boolean data item.
    h. For the maximum permissible number of data items that can be specified with data-name-1, see Appendix B, "System Quantitative Restrictions".
5. File-name-2, file-name-3, and file-name-4 must be defined in a file description entry other than the sort-merge file description entry in the DATA DIVISION.
6. Two files specified in a single MERGE statement may not exist on the same set of file reels.
7. The same file may not be named by file-name-1 to file-name-4.
8. More than one set of file-name-1 to file-name-4 may not be specified in a single SAME AREA clause or in a single SAME SORT/SORT-MERGE AREA clause, though more than one set of file-name-4 may appear in a single SAME RECORD AREA clause.
9. THROUGH and THRU are synonymous.
10. If file-name-4 is a indexed file, it must conform to the following rules:
    a. The first KEY phrase to appear must be an ASCENDING KEY phrase.
    b. The inter-record character position of the data item specified with the first data-name-1 must be equal to that of the data item associated with the prime record key of the indexed file. The two data items must be of the same length.

11. File-name-4 must conform to the following rules:

    a. If the file identified by file-name-4 has a variable-length record format, the record length of the file identified by file-name-1 must be greater than or equal to the minimum record length of the file identified by file-name-4 but must not exceed the maximum record length.

    b. If the file identified by file-name-4 has a fixed-length record format, the record length of the file identified by file-name-1 must not exceed that of file-name-4.

12. Alphabet-name-1 may not be associated with any of the following in the ALPHABET clause in a SPECIAL-NAMES paragraph.

    - function-name
    - SJIS
    - UTF8
    - UTF16
    - UTF16LE
    - UTF16BE
    - UTF32
    - UTF32LE
    - UTF32BE

    Specifying SJIS, UTF8, UTF16, UTF16LE, UTF16BE, UTF32, UTF32LE or UTF32BE in the ALPHABET clause is specific to [Win32], [Winx64] and [Linux64].

F

13. The CHARACTER TYPE clause or PRINTING POSITION clause may not be specified in the record description entries of file-name-1 to file-name-4.

14. File-name-2 to file-name-4 must name any one of the following files:

    a. A sequential file on a mass storage or floppy disk device,

    b. A sequential access method relative file, or

    c. A sequential access method indexed file.

15. For the maximum permissible number of files that can be named by file-name-2 and file-name-3, see Appendix B, "System Quantitative Restrictions".

General Rules

Rules Governing the Coding of the MERGE Statement as a Whole

1. The records of file-name-2 and file-name-3 must be arranged in the order as specified in the ASCENDING KEY or DESCENDING key phrase. If the records are not arranged in the specified order, the validity of the result of execution of the MERGE is undefined.

2. Executing a MERGE statement causes the following operations to be executed:

    a. A READ statement is executed for the files identified by file-name-2 and file-name-3, with the record read by the READ statement being passed to the merge module. The files identified by file-name-2 and file-name-3 must not be open before the start of execution of this stage. The files identified by file-name-2 and file-name-3 open and close automatically during this stage. This process is called the first stage of a merge operation.

    b. All the records in the files identified by file-name-2 and file-name-3 are merged. This process is called a merge operation.

    c. The following processes make the records thus merged available for use as records for the file identified by file-name-1:

    This process is called the last stage of a merge operation.

- If the OUTPUT PROCEDURE phrase is specified, an output procedure is executed. An output procedure is one written in procedure-name-1 or in procedure-name-1 through procedure-name-2. If the RETURN statement is executed for the file identified by file-name-1 in the output procedure, the records in the file identified by file-name-1 become available for use.

- If the GIVING phrase is specified, the merged records become available for use as those in file-name-1 and are written to the file named by file-name-4. The file identified by file-name-4 must not already be open before the start of execution of this stage. The file identified by file-name-4 opens and closes automatically during this stage.

3. In the first stage of the merge operation, the following processes are executed in sequence:

   a. Initialization processing is executed. Initialization is equivalent to the execution of an OPEN statement with the INPUT phrase specified for the files identified by file-name-2 and file-name-3. If the OUTPUT PROCEDURE phrase is specified, initialization takes place before control passes to the output procedure.

   b. The records in the files identified by file-name-2 and file-name-3 are passed to the merge operation. This is equivalent to the execution of a READ statement with the NEXT and AT END phrases specified for the files identified by file-name-2 and file-name-3 until there are no more records to read.

   c. Termination processing is executed. This is equivalent to the execution of a CLOSE statement with only file-name-2 or file-name-3 specified. If the OUTPUT PROCEDURE phrase is specified, termination is not executed until control has passed to the last statement in the output procedure.

   It may happen during the processes (a) through (c) above that a USE AFTER STANDARD EXCEPTION procedure relating to the files identified by file-name-2 to file-name-4 is executed. Statements that manipulate any of the files identified by file-name-2 to file-name-4 or that reference records in these files may not be executed during the USE procedure.

4. If the file identified by file-name-1 has a fixed-length record format and file-name-2 and file-name-3 have a record format shorter than that of the file identified by file-name-1, the vacant positions on the right side of each record are padded with blanks when the record is passed to the file identified by file-name-1.

5. The data item identified by data-name-1 is called a key item. When two or more key items are specified, they are arranged from left to right in the order of descending key strength.

   a. If the DESCENDING phrase is specified, the values of the key items of the records in the files identified by file-name-2 and file-name-3 are compared according to the rules of comparison and are merged in the order of ascending key item values.

   b. If the DESCENDING phrase is specified, the values of the key items of the records in the files identified by file-name-2 and file-name-3 are compared according to the rules of comparison and are merged in the order of descending key item values.

6. If, as a result of comparison in (5), two or more records are found in which all the key items have the same contents, the records in the files identified by file-name-2 and file-name-3 are returned in the order of their entry in the file list. If such multiple records exist in the single file identified by either file-name-2 or file-name-3, the records are returned before they are returned from any other file.

7. Where the rules of nonnumeric comparison apply, the collating sequence of the key items is determined by the following rules:

   a. If the COLLATING SEQUENCE phrase is specified, the collating sequence of characters associated with alphabet-name-1 governs.

   b. When the COLLATING SEQUENCE phrase is omitted, if the PROGRAM COLLATING SEQUENCE clause is specified in the SOURCE-COMPUTER paragraph in the IDENTIFICATION DIVISION, the collating sequence of the characters specified in that clause governs. If this clause is omitted, the computer-specific character collating sequence governs.

Rules for OUTPUT PROCEDURE Phrase

1. If the OUTPUT PROCEDURE phrase is specified, control passes to the output procedure upon completion of the merge operation. Control passes further to the executable statement next to the MERGE statement after execution of the last statement in the output procedure.

2. When control passes to the output procedure, the records in the file identified by file-name-1 have been completely merged and are ready for return in sequence by a RETURN statement, which is written in the output procedure in association with the file identified by file-name-1. The RETURN statement returns one record from the last stage of the merge operation.
3. The statements that are executed in the output procedure are called the scope of the output procedure. The scope of the output procedure covers the following:
    a. Statements in the output procedure
    b. All statements that are executed as the result of a transfer of control by executing statements in the range of output procedure (e.g. CALL statement)
    c. All the statements in the declarative procedure that are executed by any statement written in the output procedure
4. A MERGE, RELEASE, or SORT statement may not be executed in the output procedure scope.
5. The EXIT PROGRAM statement specified in the same procedure division as that of the MERGE statement cannot be executed in the range of an output procedure.

Rules for GIVING Phrase

1. If the GIVING phrase is specified, all the records in the file named by file-name-1 are moved to the file named by file-name 4 upon completion of the merge operation.
2. The file identified by file-name-4 must not already be open before the start of execution of the MERGE statement.
3. The following processes are performed in sequence on the file named by file-name-4:
    a. Initialization processing is executed. Initialization is equivalent to the execution of an OPEN statement with the OUTPUT phrase specified for the file named by file-name-4.
    b. The records in the file identified by file-name-1 are returned from the merge operation. This is equivalent to the execution of a WRITE statement with only a record name specified for the file identified by file-name-1 until there are no more records to write.

       If a relative file is identified by file-name-4, the relative record numbers of the records that are returned are assigned in sequence beginning with the first record as 1, 2, 3, and so on. If a relative key item is specified in the file identified by file-name-4, the relative key item is set to a relative record number each time a record is returned. The relative key item, after the execution of the MERGE statement, points to the last record returned to the file identified by file-name-4.
    c. Termination processing is executed. This is equivalent to the execution of a CLOSE statement with only file-name-4 specified.

    It may happen during the processes (a) through (c) above that a USE AFTER STANDARD EXCEPTION procedure relating to the file identified by file-name-4 is executed. Statements that manipulate the file identified by file-name-4 or that reference records in that file may not be executed during the USE procedure.
4. When an attempt is made for the first time to write a record beyond the range defined externally to the file identified by file-name-4, either of the following processes is executed:
    a. If a USE AFTER STANDARD EXCEPTION procedure relating to the file identified by file-name-4 is written, that USE procedure is executed. After control returns from the USE procedure, termination processing is executed as if a CLOSE statement with only file-name-4 specified were executed.
    b. If a USE AFTER STANDARD EXCEPTION procedure relating to the file identified by file-name-4 is not written, termination processing is executed as if a CLOSE statement with only file-name-4 specified were executed.
5. If the file identified by file-name-4 has a fixed-length record format and file-name-1 has a record format shorter than that of the file identified by file-name-4, the vacant positions in the right side of each record are padded with blanks when the record is returned to the file named by file-name-4.

6.4.28 MOVE Statement (Nucleus)

Moves data from one data item to another.

For details about how to use the examples, refer to "MOVE Statement with CORRESPONDING" in the "Syntax Samples".

Format 1

Move data to one or more data items.

```
MOVE { identifier-1 } TO {identifier-2}...
     { literal-1    }
```

Format 2

Move a data item that is subordinate to two group items by maintaining its correspondence.

```
MOVE { CORRESPONDING } identifier-1 TO identifier-2
     { CORR          }
```

F

Format 3

Move one data item to one data item with the encoding form conversion.

```
MOVE { CONVERSION } identifier-3 TO identifier-4
     { CONV       }
```

Format-3 is specific to [Win32], [Winx64] and [Linux64].

Syntax Rules

Rules Common to Format 1 and Format 2

1. CORRESPONDING and CORR are synonymous.
2. Identifier-1 and identifier-2 in format 2 must each be a group item.
3. Identifier-1 and identifier-2 in format 2 may not be an index data item.

F

4. When identifier-2 is a strongly typed group item, literal-1 cannot be specified.
5. When identifier-2 is a strongly typed group item, identifer-1 must also be a group item that is strongly typed with the same type.

F

Rule for Format 3

1. CONVERSION and CONV are synonymous.
2. Identifier-3 and Identifier-4 must be alphanumeric data items or national data items. In addition, they can't match the following items.
   - Reference modification
   - Function Identifier
   - In-line invocation
   - Object property
3. Identifier -3 and Identifier -4 can't match the item that contains the following clauses.
   - JUSTIFIED clause
   - ANY LENGTH clause

4. Identifier-3 and identifier-4 must not have the same encoding form.
5. The sending side operand and receiving side operand must not match with items that partially or entirely share the same storage.

General Rules

Rules Common to Format 1 and Format 2

1. If a variable occurrence data item is specified as identifier-1 or identifier-2, the evaluation of the length of the variable occurrence data item is influenced by the value of the data item specified in the DEPENDING ON phrase. For more details on the evaluation of the length of a variable occurrence data item, see the topic titled "OCCURS Clause" in the "Data Description Entry" section of Chapter 5, the "Data Division".
2. For more details on the valid combination of sending and receiving operands and the operation of the MOVE statement, see the topic titled "Rules for Transcribing Data" in the "Common Statement Rules" section of earlier in this chapter.

Rules for Format 1

1. The format 1 MOVE statement copies the contents of the operand specified before the TO phrase to each operand specified after the TO phrase. The operand specified before the TO phrase is called the sending operand, and each operand written after the TO phrase is called the receiving operand.
2. If identifier-1 is subscripted or reference-modified or if a function-identifier is specified identifier-1, the subscript, reference identifier, and function-identifier are evaluated only once before the sending operand is moved to the starting receiving operand.
3. If identifier-2 is subscripted or reference-modified, the subscript and reference identifier are evaluated once just before the sending operand is moved to each receiving operand.
4. The length of the data item specified by identifier-1 is evaluated once just before the sending operand is moved to the receiving operands.
5. The length of the data item specified by identifier-2 is evaluated just before it is moved to each receiving operand.

Rule for Format 2

The format 2 MOVE statement copies the contents of data items that are subordinate to identifier-1 and identifier-2 to maintain similarity in their qualifiers; that is, it transcribes the data item subordinate to identifier-1 as a sending data item and the data item subordinate to identifier-2 as a receiving data item. The format 2 MOVE statement is equivalent to the execution of a separate MOVE statement for each corresponding identifier.

F

Rules for Format 3

1. In the MOVE statement of format 3, the sending side operand is moved after converting to the encoding form of the receiving side.
2. In case of a character string that includes the trailing space in identifier-3, the character string without the trailing space becomes the target of conversion. While moving the result of the conversions, trailing spaces are inserted to match the encoding of the receiver side. However, the trailing space is not inserted when the conversion result exceeds the size of the receiving side.
3. When the conversion result exceeds the size of the receiving side, the characters that completely fit into the receiving side are moved.
4. The following two types of special registers are being auto-generated.
    a. CONV-STATUS

    CONV-STATUS is used as a numeric item defined as "PIC S9(4) COMP-5". It is used for reference, whether code conversion is a success or not.

    The initial value of CONV-STATUS is zero. It is automatically initialized when format 3 is executed.

    When executing the MOVE statement of return code value format 3, the return code is set in CONV-STATUS. The value of the return code is any of the following.

- 0: the code conversion is a success.

- 1: the substitution character is included in the code conversion results.

- 2: there was a failure in the code conversion.

b. CONV-SIZE

CONV-SIZE is used as a numeric item defined as "PIC S9(4) COMP-5".

It is used for reference if there is any issue in the code conversion size .

The initial value of CONV-SIZE is zero.

It is automatically initialized at the beginning of the execution of format 3.

When you execute the MOVE statement of return code value format 3, and when the return code of CONV-STATUS is 0 or 1, then the return code is set in CONV-SIZE.

The return code value is any of the following.

- 0: the result of code conversion fits into the operand on the receiving side.

- 1: the result of code conversion exceeds the size of the operand on the receiving side.

Example

Example of Subscript Evaluation

```
MOVE a (b) TO b c (b)
```

The execution of the MOVE statement would yield the same result as that of the MOVE statement shown below. Temp is a temporary data item.

```
MOVE a (b) TO temp
MOVE temp TO b
MOVE temp TO c (b)
```

6.4.29 MULTIPLY Statement (Nucleus)

Calculates the result of multiplication.

Format 1

Multiplies data to one or more data items.

MULTIPLY { identifier-1 / literal-1 } BY {identifier-2 [ROUNDED]}...

[ON SIZE ERROR imperative-statement-1]

[NOT ON SIZE ERROR imperative-statement-2]

[END-MULTIPLY]

Format 2

Stores the result of multiplication in a data item different from the multiplier.

```
MULTIPLY { identifier-1 } BY { identifier-2 }
         { literal-1   }     { literal-2   }
         GIVING {identefier-3 [ROUNDED]}...
         [ON SIZE ERROR imperative-statement-1]
         [NOT ON SIZE ERROR imperative-statement-2]
         [END-MULTIPLY]
```

Syntax Rules

1. Identifier-1 and identifier-2 must each be a numeric data item.
2. Identifier-3 must be a numeric edited data item.
3. Literal-1 and literal-2 must each be a numeric literal.

General Rules

Rule for Format 1

The MULTIPLY statement in format 1 multiplies the operand specified before the BY phrase by identifier-2 and stores the result in identifier-2. It executes multiplication in the order of identifiers-2 specified.

Rule for Format 2

The MULTIPLY statement in format 2 multiplies the operand specified before the BY phrase by the operand specified after the BY phrase and stores the result in identifier-3. It executes multiplication in the order of identifiers-3 specified.

Rules Common to Format 1 and Format 2

1. The END MULTIPLY phrase delimits the scope of the MULTIPLY statement.
2. For more details on the ROUNDED phrase, ON SIZE ERROR phrase, see the section titled "Common Statement Rules" earlier in this chapter and the topic titled "Rules for Transcribing Data" in the "Common Statement Rules" section earlier in this chapter.

6.4.30 OPEN Statement (Sequential File, Relative File, Indexed File)

The OPEN statement opens a file and makes it available for the program.

The RESERVE phrase in the OPEN statement of the sequential file is an obsolete element.

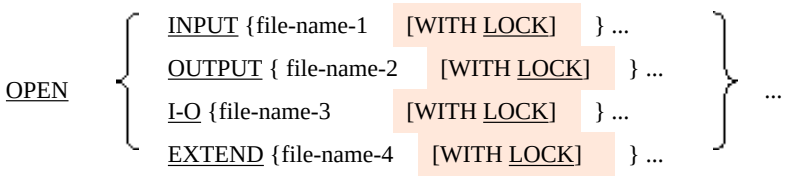
Format 1

Sequential file

```
                                  [ REVERSED                                  ]  }
                                  [                                           ]  }
OPEN  { INPUT  { file-name-1      [ WITH { LOCK     } ]                       ]  } ...
      {                           [      { NO REWIND } ]                       ]  }
      {                                                                          }
      { OUTPUT { file-name-2      [ WITH { LOCK     } ]                       ]  } ...
      {                           [      { NO REWIND } ]                       ]  }
      { I-O file-name-3   [WITH LOCK]  } ...                                     }
      EXTEND file-name-4        [WITH LOCK]  } ...
```

Format 2

Relative file, indexed file



Syntax Rules

Rule Common to All File Types

Files with different ORGANIZATION clauses or different ACCESS MODE clauses can be specified for file-name-1 to file-name-4.

Rules for Sequential Files

1. When the REVERSED or NO REWIND phrase is specified, file-name-1 must be a record sequential file.
2. F The WITH NO REWIND clause is regarded as a comment.
3. When the EXTEND phrase is specified, file-name-4 must be a file without the LINAGE phrase. However, the EXTEND phrase cannot be specified on a multiple file reel.
4. F When the INPUT phrase is specified, file-name-1 cannot be a print file.
5. When the I-O phrase is specified, file-name-3 cannot be a line sequential file or print file.

Rules for Relative Files and Indexed Files

When the EXTEND phrase is specified, file-name-1 must be a sequential access file.

General Rules

Rules Common to All File Types

File-name-n in the following description denotes file-name-1 to file-name-4.

1. When the OPEN statement has executed successfully, a file assigned to file-name-n can be used in the program and the file is open. When the OPEN statement has executed successfully, the file is related to file-name by means of a file connector.
2. When the file assigned to file-name-n physically exists and is recognized by the I-O-control system, "the file is usable". If not, we say, "the file is not usable". Results of OPEN statement execution in each status are listed below.

OPEN statement specification	File usable	File unusable
INPUT	The file is open.	The OPEN statement fails.
INPUT (optional file)	The file is open.	The file is open. The first read leads to file termination or an invalid key condition.
I-O	The file is open.	The OPEN statement fails.
I-O (optional file)	The file is open.	The file is created by executing the OPEN statement.
OUTPUT	The file is open. The file contains no records.	The file is created by executing the OPEN statement.
EXTEND	The file is open.	The OPEN statement fails.
EXTEND (optional file)	The file is open.	The file is created by executing the OPEN statement.

3. When the OPEN statement has executed successfully, the record area of file-name-n becomes usable.
4. An input-output statement for the file must not be executed before opening the file.

5. The file open mode is specified by the INPUT, OUTPUT, I-O, and EXTEND phrases. The input-output statement to be executed after the file is opened depends on the open mode. See the topic titled "Operation of Input-Output Statements" of the "Input-Output Facility" section of Chapter 2, "COBOL Modules" for details of the relationship between the open mode and input-output statement.
6. The first record is not written or read by the OPEN statement.
7. When one run unit contains more than one execution of the OPEN statement for one file, a CLOSE statement must be executed before each subsequent execution. This CLOSE statement must not have a REEL, UNIT (sequential file only), or LOCK phrase.
8. If a file attribute conflict condition occurs during the OPEN statement execution, the statement execution fails and the following sequence of processing is performed:
    a. The I-O status for file-name-n is set.
    b. If a procedure for the USE AFTER STANDARD EXCEPTION related to file-name-n is written, the procedure will be executed. After execution, the OPEN statement is controlled according to the rule for the USE statement. If file-name-n has a FILE STATUS clause specified and there is no USE AFTER STANDARD EXCEPTION procedure associated with the file, control transfers to the end of the OPEN statement. If there is no USE AFTER STANDARD EXCEPTION or FILE STATUS associated with the file, the execution results are undefined.
9. When the OPEN statement is executed, the value of the I-O status in file-name-n is updated.
10. When an I-O phrase is written, the file assigned to file-name-3 must be a file that can execute both input and the output statements. When the OPEN statement in the I-O phrase has executed successfully, other input and output statements can be executed.

F

11. The file lock mode is specified in the LOCK phrase. The file lock mode is determined by the descriptions of the LOCK MODE clause and the OPEN statement in the file control entry. See the topic titled "Sharing and Exclusion of Files" in the "Input-Output" section of Chapter 2, "COBOL Modules" for details on the file lock mode.
12. When the LOCK phrase is specified, the file assigned to file-name-n is locked by executing the OPEN statement, which is the same as specifying LOCK MODE EXCLUSIVE in the file-control entry of file-name-n.
13. When the OPEN statement in exclusive mode has executed successfully, the file related to the file connector of file-name-n is exclusively locked. If the file assigned to file-name-n has already been opened by another file connector, the OPEN statement execution fails.
14. After the OPEN statement in shared mode has executed successfully, the file assigned to file-name-n can be opened by one or more file connectors. If the file assigned to file-name-n has already been opened by another file connector, the OPEN statement execution is successful.
15. When an OPEN statement with the INPUT phrase is first executed for a file that does not exist, the file position indicator is set to indicate that no file exists.
16. When an OPEN statement with the EXTEND or I-O phrase for a file that does not yet exist executes successfully, a file is created. The file creation is performed as if two statements have been executed in the following sequence:
    a. OPEN OUTPUT file-name-n
    b. CLOSE file-name-n

    The created file is empty and the file position indicator is set to one.
17. The execution results of an OPEN statement with two or more file-name-n specified is equivalent to executing individual OPEN statements for each file-name-n in the sequence written.
18. The minimum and the maximum sizes of the record in the file are determined according to the user specification when the file is created. These sizes cannot be changed afterwards.

Rules for Sequential Files

1. If an OPEN statement with the INPUT or I-O phrase has executed successfully, the file position indicator of file-name-n is set to one.

2. If an OPEN statement with the EXTEND has executed successfully, the file is positioned just after the last logical record. The last logical record of a sequential file is the last record written in the file.
3. When a record sequential file with the LABEL RECORDS phrase is specified as file-name-n, the header label is processed as follows:
    a. When an OPEN INPUT statement has been executed, the label is tested according to the standard label rules.
    b. When an OPEN OUTPUT statement has been executed, the label is written according to the standard label rules.

    If there is no label even though the LABEL RECORDS phrase exists, or there is a label but no LABEL RECORDS phrase exists, the result of OPEN statement execution is undefined.
4. When an OPEN statement with the EXTEND phrase has been executed for a record sequential file with the LABEL RECORDS phrase, the label is processed as follows:
    a. The header label is processed only when the file is a sequential single reel/unit file.
    b. When the file is a sequential multiple/unit file, the header volume label of the last reel/unit is tested according to the standard label rules.
    c. The ending file label is tested according to the standard label rules. After testing, this label is deleted.
    d. Then, the label is written according to the standard label rules.
5. When an OPEN statement with the I-O phrase has been executed for a record sequential file with the LABEL RECORDS phrase, the label is processed as follows:
    a. The label is tested according to the standard label rules.
    b. The label is written according to the standard label rules.
6. A multiple tape file is regarded to be the same as a sequential single tape file.
7. When assigning a multiple reel file to file-name-n, the following rule is applied.
    a. No more than one file can be opened on a one tape reel at one time.
    b. When an OPEN statement with the INPUT phrase is executed, the position number of file-name-n need not be more than the position number of a file that has been opened before.
    c. When an OPEN statement with the OUTPUT phrase is executed, the position number of file-name-n must not be less than the maximum number of the files on the related tape reel or all subsequent files will be lost.
    d. File-name-n must be a sequential file.
8. An OPEN statement which omits the NO REWIND or REVERSED phrases can be executed only for files as follows:
    a. Sequential single reel/unit file
    b. Sequential file under environment of multiple file tapes completely included in one tape reel

    See the topic titled "MULTIPLE FILE TAPE clause (Sequential File and Report Writer)" of the "Input-Output Section" section in Chapter 4, the "Environment Division" for details on the multiple file tape environment.
9. When the NO REWIND or REVERSED phrase is specified in a file assigned to file-name-n that cannot apply either function (such as a mass storage file), these phrases are ignored.
10. When a rewindable unit is assigned to a file-name-n file, the following rules are applied.
    a. If an OPEN statement has been executed and the REVERSED, EXTEND, and NO REWIND phrases are omitted, the file is positioned to the beginning of the file.
    b. When the NO REWIND phrase is specified, the beginning of the file must be specified before executing the OPEN statement. When the OPEN statement has been executed successfully, the file is positioned to the specified position.

c. When the REVERSED phrase is specified, the file is positioned at the end of the file if the OPEN statement has been successfully executed.

11. When the REVERSED phrase is specified, records can be used in the reversed order from the last record if a READ statement for the file assigned to file-name-n has been executed after executing the OPEN statement.
12. No label processing is performed for line sequential or print files.
13. Print files must not be opened with the INPUT or I-O phrases.
14. When an OPEN statement for a sequential file has been executed successfully, the volume indicator is set as follows:

    a. When an OPEN statement with the INPUT or I-O phrase for usable files has been executed, the volume indicator is set to indicate the first reel/unit or single reel/unit.

    b. When an OPEN statement with the EXTEND phrase has been executed, the volume indicator is set to indicate the reel/unit containing the last record.

    c. When an OPEN statement with the OUTPUT or I-O phrase for unusable files has been executed, the volume indicator is set to indicate a new reel/unit.

Rules for Relative Files

1. When an OPEN statement with the INPUT or I-O phrase has been executed successfully, the file position indicator of file-name-n is set to 1.
2. If an OPEN statement with the EXTEND phrase has been executed successfully, the file is positioned just after the last record. The last record of a relative file has the highest relative record number among existing records.

Rules for Indexed Files

1. When an OPEN statement with the INPUT or I-O phrase has been executed successfully, the file position indicator of file-name-n is set to the minimum value that can be assigned to the prime record key of the file. The prime record key is set as a key of reference.
2. If an OPEN statement with the EXTEND phrase has been executed successfully, the file is positioned just after the last record. The last record of an indexed file has the highest prime record key value among existing records.

F However, if the DUPLICATES phrase is specified in the RECORD KEY clause, the last record of the indexed file is the last record created among existing records with the maximum prime record key value.

F 6.4.31 OPEN Statement (Presentation File)

The OPEN statement makes a Presentation file usable.

Format

```
OPEN { INPUT {file-name-1}...  }
     { OUTPUT {file-name-2}... } ...
     { I-O {file-name-3}...    }
```

Syntax Rules

In addition to Presentation files, sequential files, relative files, and indexed files can be specified in a list of file-name-1, file-name-2, or file-name-3. Files with different ORGANIZATION or ACCESS MODE clauses can also be specified in the list.

General Rules

In the following general rules, file-name-n refers to file-name-1, file-name-2, and file-name-3:

1. When an OPEN statement has been executed successfully, a file assigned to file-name-n can be used in the program and the file is open. When an OPEN statement has been executed successfully, the file is related to file-name-n by means of the file connector.
2. When an OPEN statement has been executed successfully, the record area of file-name-n becomes usable.
3. Input-output statements for the file must not be executed before opening the file.
4. The file open mode is specified with the INPUT, OUTPUT, I-O, and EXTEND phrases. The input-output statements that can be executed after the file is opened depend on the open mode. See the topic titled "Operation of Input-Output Statements" in the "Input-Output Facility" section of Chapter 2, "COBOL Modules" for details of the relationship between the open mode and input-output statement.
5. The open mode specified for file-name-n is dependant on the destination type that is specified in the SYMBOLIC DESTINATION clause of file-name-n. The following table shows the relationship between destination type and open mode.

Destination Type	Open Statement Specification		
	INPUT	OUTPUT	I-O
ACM	o	o	o
DSP	-	-	o
PRT	-	o	o
APL	o	o	o
TRM	-	o	o

   o: Executable

   -: Un-executable

6. The first record is not written or read by the OPEN statement.
7. When an OPEN statement has been executed, the value of the I-O status of file-name-n is updated.
8. The execution results of an OPEN statement with two or more file-name-n specified is equivalent to executing individual OPEN statements for each file-name-n in the sequence written.

6.4.32 OPEN Statement (Report Writer)

The OPEN statement makes a Report Writer file usable.

Note

The report writer module cannot be used in [Winx64], [LinuxIPF] and [.NET].

Format

```
OPEN { OUTPUT {file-name-1 [WITH NO REWIND ]}... } ...
     { EXTEND {file-name-2}...                    }
```

Syntax Rule

Only the OUTPUT or EXTEND phrases can be specified by an OPEN statement for a report writer file.

General Rules

1. The OPEN statement for a report writer file must be executed for the report related to the file before executing the INITIALIZE statement.

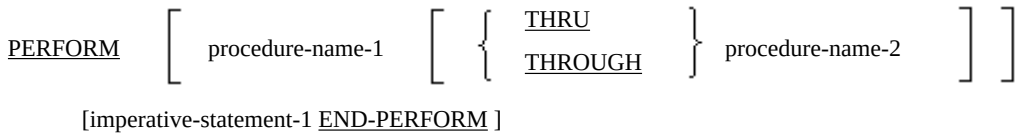
2. The rules for each clause of an OPEN statement for a report file are the same as an OPEN statement for sequential files. See the topic titled "OPEN Statement (Sequential file, Relative file, Indexed file)" earlier in this section.

6.4.33 PERFORM Statement (Nucleus)

The PERFORM statement executes a series of statements repeatedly.

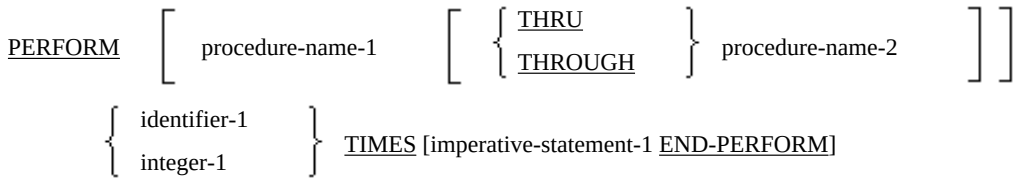
Format 1

A group of statements is executed only once.



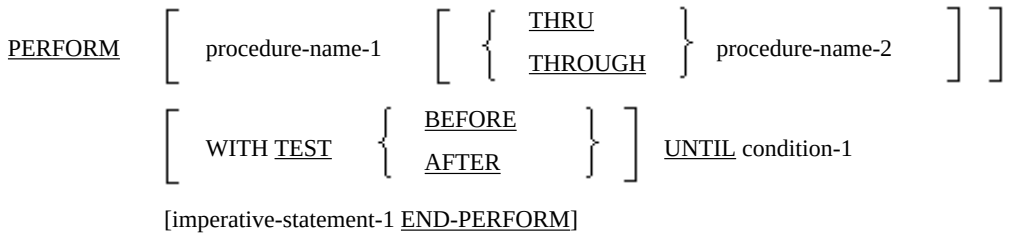
Format 2

A group of statements is executed as many times as specified.



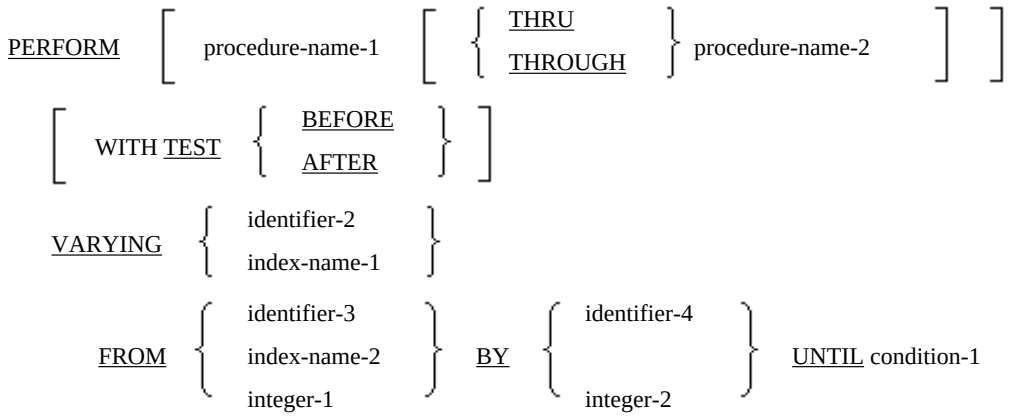
Format 3

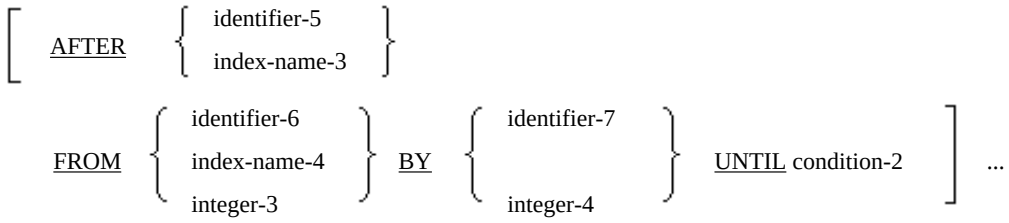
A group of statements is executed repeatedly until conditions are satisfied.



Format 4

The value of a data item or index-name is changed according to a number of executions while a group of statements is executed repeatedly until conditions are satisfied.





[imperative-statement-1 END-PERFORM]

F

Format 5

A group of statements is executed repeatedly without any conditions.

PERFORM WITH NO LIMIT

imperative-statement-1 END-PERFORM

Syntax Rules

Rules Common to Formats 1 to Format 4

1. Either procedure-name-1 or imperative-statement-1 must be specified in formats 1 to 4. Procedure-name-1 and imperative-statement-1 cannot both be specified.
2. THROUGH is synonymous with THRU.
3. When specifying procedure-name-1 and procedure-name-2, if either procedure-name exists in the DECLARATIVES portion of the PROCEDURE DIVISION, the other procedure-name must also exist in the same DECLARATIVES section.

Rules Common to Formats 3 and Format 4

1. When the TEST BEFORE and TEST AFTER phrases are omitted in formats 3 and 4, the TEST BEFORE phrase is assumed.
2. The conditional expression explained in the topic titled "Conditional Expression" in the "Common Statement Rules" section earlier in this chapter should be specified in condition-1 and condition-2 in formats 3 and 4.

Rules for Format 2

1. Identifier-1 must be an integer item.
2. See Appendix B, "System Quantitative Restrictions" for the maximum value that can be specified for integer-1 and identifier-1.

Rules for Format 4

1. When imperative-statement-1 is specified, the AFTER phrase must not be used.
2. Identifier-2 to identifiers-7 must be integer items.
3. Literal-1 to literal-4 must be numeric literals.
4. Literal-2 and literal-4 must not be zero.
5. When index-name-1 is specified in the VARYING phrase, operands for the FROM phrase and the BY phrase must conform to the following rules:
    a. Identifier-3 and identifier-4 must be integer items.
    b. Literal-1 must be a positive integer.
    c. Literal-2 must be a non-zero integer.
6. When index-name-3 is specified in the AFTER phrase, operands for the FROM phrase and the BY phrase must conform to the following rules:
    a. Identifier-6 and identifier-7 must be integer items.

b. Literal-3 must be a positive integer.

c. Literal-4 must be a non-zero integer.

7. When index-name-2 is specified in the FROM phrase in the VARYING phrase, operands for the VARYING phrase and the BY phrase must conform to the following rules:

   a. Identifier-2 and identifier-4 must be integers.

   b. Literal-2 must be a non-zero integer.

8. When index-name-4 is specified in the FROM phrase in the AFTER phrase, operands for the VARYING phrase and the BY phrase must conform to the following rules:

   a. Identifier-5 and identifier-7 must be integers.

   b. Literal-4 must be a non-zero integer.

9. Up to six AFTER phrases can be specified.

General Rules

Rules for All PERFORM Statements

1. A PERFORM statement specifying procedure-name-1, with or without procedure-name-2 specified, is called an "out-of-line PERFORM statement." A PERFORM statement specifying imperative-statement-1 is called an "in-line PERFORM statement." A series of statements to be executed by the PERFORM statements is referred to as a "group of statements."

2. The out-of-line PERFORM statement repeatedly executes the group of statements written in another section or paragraph the number of specified times or until the condition is satisfied.

3. The in-line PERFORM statement repeatedly executes the group of statements (imperative-statement-1) written in the in-line PERFORM statement for the specified times or until the condition is satisfied

4. Other rules apply equally to the out-of-line PERFORM statement and the in-line PERFORM statement.

Rules for a Group of Statements Executed in the PERFORM Statement

1. The PERFORM statement logically consists of statements to be executed until control returns at the end of the PERFORM statement. When a CALL, an EXIT, a GO TO, or a PERFORM statement is written in the group of statements, all statements to be executed by these statements are included in the range of the PERFORM statement. Out of line statements included in the range need not be written serially in the program.

2. Statements executed as the result of a transfer of control caused by executing an EXIT PROGRAM statement within a PERFORM are not considered to be part of the range of the PERFORM statement.

Rules for a Group of Statements Executed by an In-line PERFORM Statement

1. The in-line PERFORM statement repeatedly executes a group of statements specified in imperative-statement-1.

2. The END-PERFORM phrase terminates the range of the in-line PERFORM statement.

3. The group of statements executed by an in-line PERFORM statement consists of the first statement following the PERFORM statement to the last statement that appears before the END-PERFORM statement.

Rules for a Group of Statements Executed by an Out-of-line PERFORM Statement

1. The out-of-line PERFORM statement repeatedly executes a group of statements written in another section or paragraph. Statements from the first statement of the procedure-name-1 to the following statements are executed repeatedly.

   a. When procedure-name-2 has been omitted and a paragraph name is specified as procedure-name-1, the out-of-line PERFORM statement executes statements up to the last statement of the paragraph.

   b. When procedure-name-2 has been omitted and a section-name is specified as procedure-name-1, the out-of-line PERFORM statement executes statements up to the last statement of the section.

   c. When a paragraph name is specified as procedure-name-2, the out-of-line PERFORM statement executes statements up to the last statement of the paragraph.

d. When a section-name is specified as procedure-name-2, the out-of-line PERFORM statement executes statements up to the last statement of the section.

2. The relationship between procedure-name-1 and procedure-name-2 specifies the flow of control, starting from procedure-name-1 and ending at procedure-name-2. A GO TO or PERFORM statement can be written anywhere between procedure-name-1 and procedure-name-2. To return control from the PERFORM statement which is not the last statement in the group of statements, a paragraph containing only the EXIT statement must be defined to return control from the paragraph containing only the EXIT statement to the PERFORM statement. Procedure-name-2 must be a paragraph name of a paragraph containing only the EXIT statement or a section-name of a section containing a paragraph consisted of only the EXIT statement.

3. 3. A group of statements can be executed by a statement other than the PERFORM statement. When control transfers from a statement other than the PERFORM statement to a group of statements, control does not return to the PERFORM statement from the last statement of the group of statements. Instead, control transfers to the next executable statement after the last statement in the group of statements.

4. Procedure-name-1 and procedure-name-2 must be section-names or paragraph names in the program in which the PERFORM statement has been written. A procedure-name in the program containing the program in which the PERFORM statement has been written or a procedure-name in the program contained in the program in which the PERFORM statement has been written cannot be specified as procedure-name-1 or procedure-name-2.

5. The following rules apply when writing another PERFORM statement in the range of a PERFORM statement. The former PERFORM statement is called "containing PERFORM statement" and the latter is called "contained PERFORM statement."

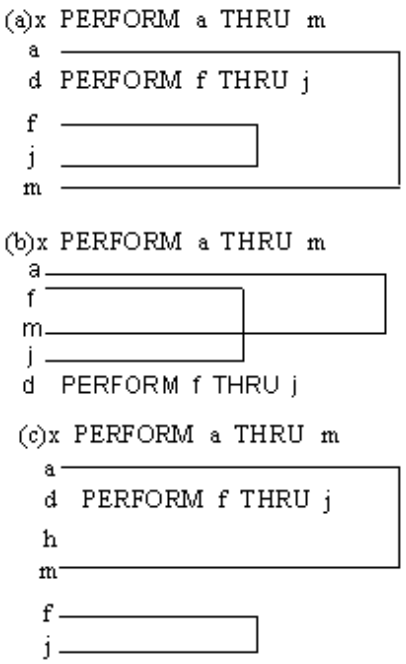
   a. The range of the contained PERFORM statement must be either completely within or completely outside the range of logical execution of the containing PERFORM statement. That is, the exit of the containing PERFORM statement cannot be passed during execution of the contained PERFORM statement.

   b. Multiple contained PERFORM statements within the range of the containing PERFORM statements cannot have a common exit.

F

   In this implementation of COBOL, this restriction is removed.

6. The following example shows correct usage of the out-of-line PERFORM statement.



Rules for Format 1

The format 1 PERFORM statement executes a group of statements only once. Then, the control returns to the end of the PERFORM statement.

Rules for Format 2

1. The format 2 PERFORM statement repeatedly executes a group of statements as many times as specified in the initial value of integer-1 or identifier-1. Then, the control returns to the end of the PERFORM statement.
2. When the value of identifier-1 before the PERFORM statement execution is zero or negative, control transfers to the end of the PERFORM statement without executing a group of statements.
3. Even if the value of identifier-1 has been changed during execution of a group of statements, the execution number never changes.

Rules for Format 3

1. The format 3 PERFORM statement repeatedly executes a group of statements until condition-1 is satisfied. Then, the control returns to the end of the PERFORM statement.
2. To test condition-1 before executing a group of statements, specify the TEST BEFORE phrase or omit the TEST phrase. To test condition-1 after executing a group of statements, specify the TEST AFTER phrase.
3. When a subscript and a reference modifier are added to operands in condition-1, they are evaluated every time condition-1 is tested.

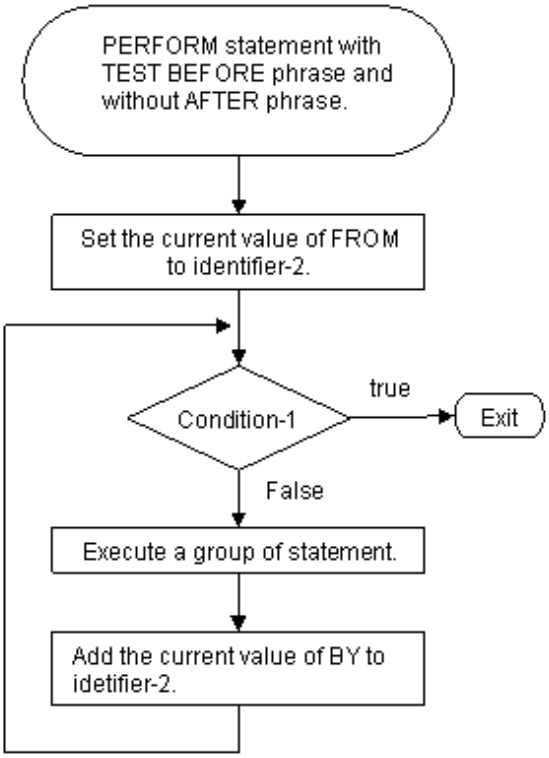
Rules for Format 4

1. The format 4 PERFORM statement repeatedly executes a group of statements until the specific condition is satisfied and changes the value of the data item according to the repetitive execution. Then, the control returns to the end of the PERFORM statement.

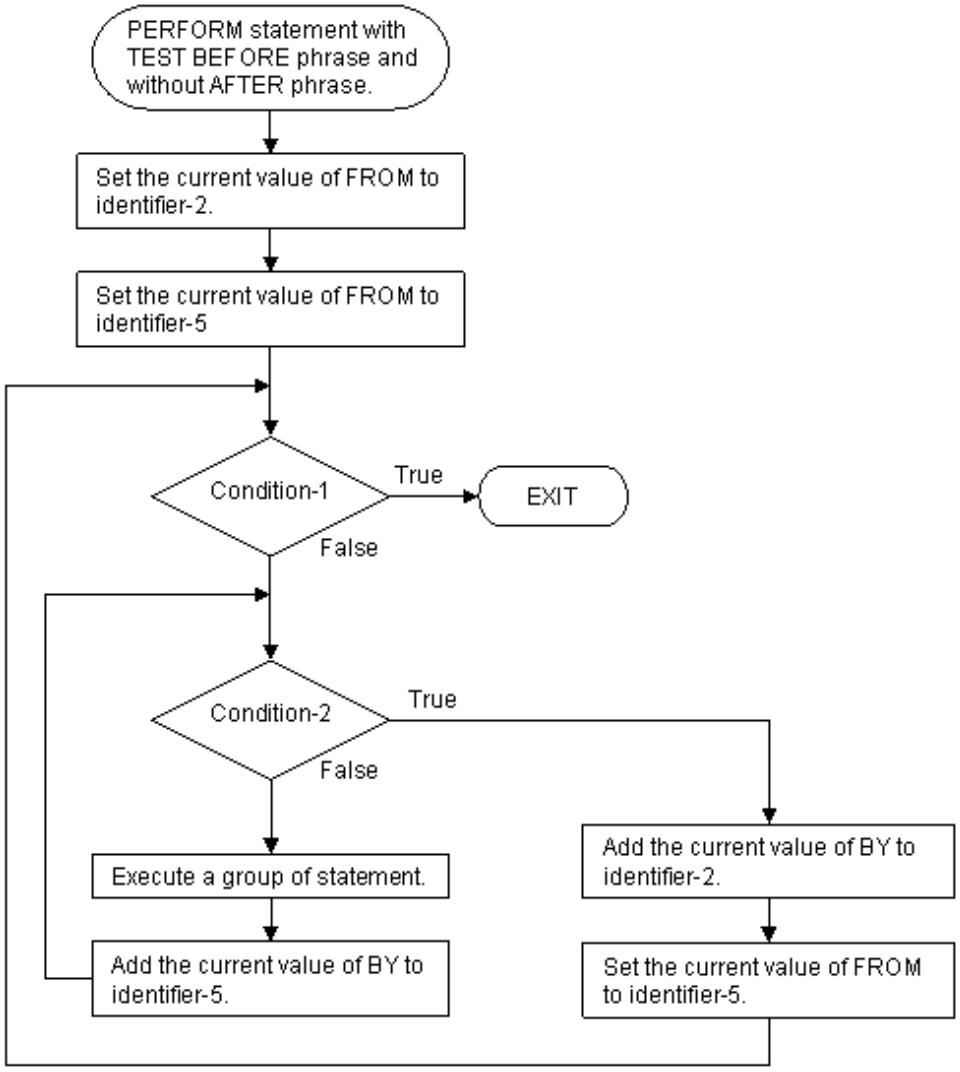
   In the out-of-line PERFORM statement, one or more data items to be changed according to the repetitive execution can be specified in the VARYING phrase and the AFTER phrase. In the in-line PERFORM statement, only one data item to be changed according to the repetitive execution can be specified in the VARYING phrase.

   Requirements for ending repetitive execution are specified after the UNTIL phrase. To test the at end condition before executing a group of statements, specify the TEST BEFORE phrase or omit the TEST phrase. To test the at end condition after executing a group of statements, specify the TEST AFTER phrase.
2. Identifier-4 and identifier-7 must not be zero.
3. When identifier-1 is specified in the VARYING phrase, the value of the operand in the FROM phrase must be a value corresponding to the occurrence number of the table containing index-name-1. The value of operand in the BY phrase must be an increment of the value corresponding to the occurrence number of the table containing index-name-1.
4. A value outside the range of the table containing index-name-1 must not be set to index-name-1 until condition-1 is satisfied. In the same way, a value outside the range of the table containing index-name-3 cannot be set to index-name-3 until condition-2 is satisfied. However, the value of index-name-1 after execution of the PERFORM statement may be as much as one increment or decrement value outside the range of the table.
5. When a subscript and a reference modifier are added to the identifier or operand in the condition of the UNTIL phrase, they are evaluated as follows:
   - Subscripts of identifier-2 and identifier-5 are evaluated every time these identifiers are set to their initial values or increment values are added.
   - Subscripts of identifier-3 and identifier-6 are evaluated when these identifiers are set to their initial values.
   - Subscripts of identifier-4 and identifier-7 are evaluated when increment values are added.
   - Reference modifiers added to condition-1 and condition-2 are evaluated every time condition-1 or condition-2 is tested.

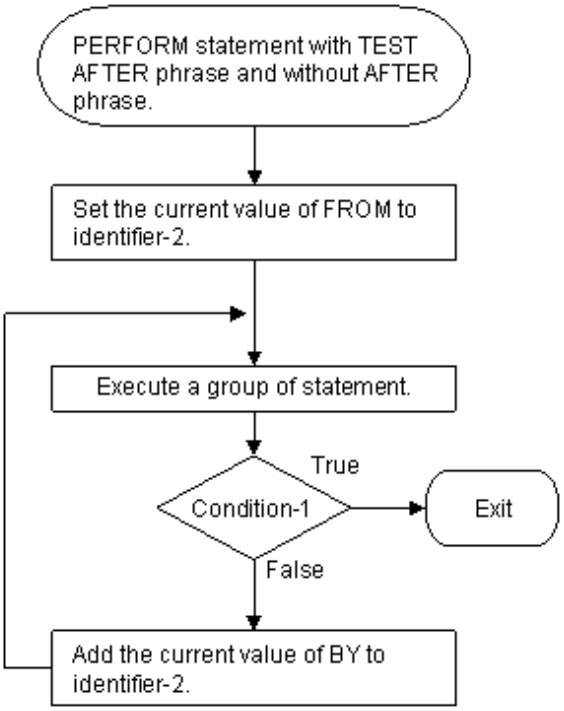
6. When the TEST BEFORE phrase is specified implicitly or explicitly, and the AFTER phrase is omitted, the processing will be as shown in the following figure.



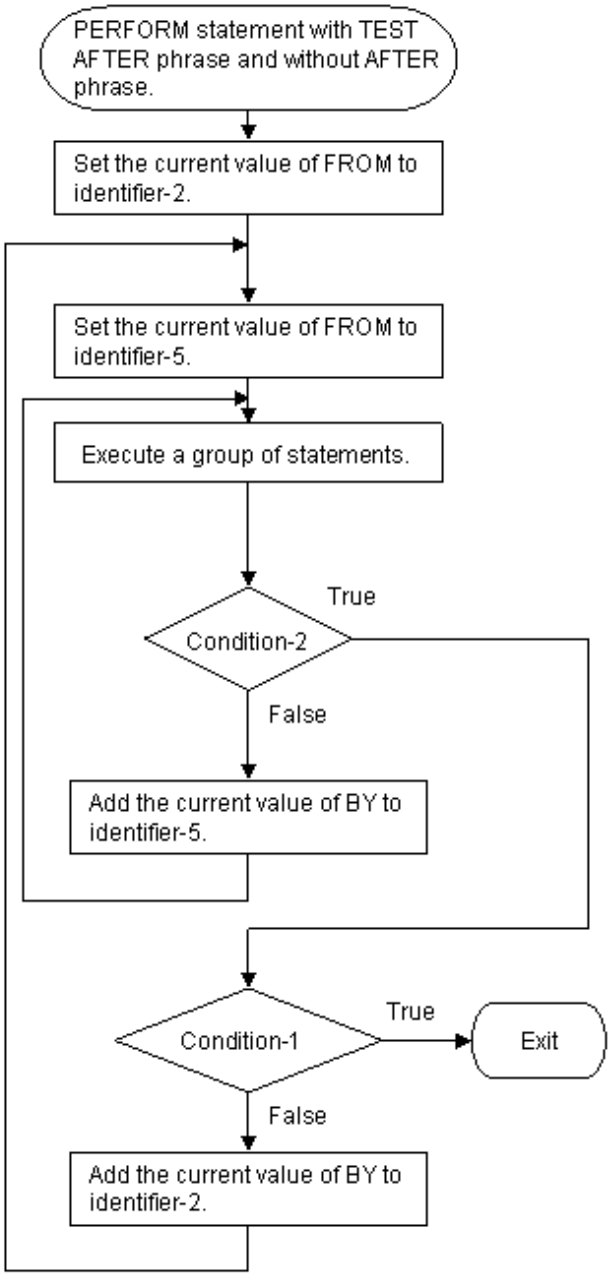
7. When the TEST BEFORE phrase is specified implicitly or explicitly, and the AFTER phrase is specified, the processing will be as shown in the following figure.



8. When the TEST AFTER phrase is specified and the AFTER phrase is omitted, the processing will be as shown in the following figure.



9. When the TEST AFTER phrase is specified and one AFTER phrase is specified, the processing will be as shown in the following figure.



10. When two or more AFTER phrases have been specified, the test of the AFTER phrase is the same as when only one AFTER phrase has been written. However, the processing for an operand in each AFTER phrase repeats every time the value of the operand in the preceding AFTER phrase is changed.

F

Rules for Format 5

The PERFORM statement in format 5 repeatedly executes a group of statements until the branch statement indicating explicit control transfer or the EXIT PERFORM statement is executed.

6.4.34 READ Statement (Sequential File, Relative File, Indexed File)

The READ statement reads a record from the file.

Format 1

Records are read sequentially. (Sequential, relative and, indexed files)

```
READ file-name-1 [NEXT] RECORD
  [INTO identifier-1]
  [WITH [NO] LOCK]
  [AT END imperative-statement-1]
  [NOT AT END imperative-statement-2]
  [END-READ]
```

Format 2

Records are read randomly. (Relative file)

```
READ file-name-1 RECORD
  [INTO identifier-1]
  [WITH [NO] LOCK]
  [INVALID KEY imperative-statement-3]
  [NOT INVALID KEY imperative-statement-4]
  [END-READ]
```

Format 3

Records are read at random. (Indexed file)

```
READ file-name-1 RECORD
  [INTO identifier-1]
  [WITH [NO] LOCK]
  [KEY IS {data-name-1} ... ]
  [INVALID KEY imperative-statement-3]
  [NOT INVALID KEY imperative-statement-4]
  [END-READ]
```

Format 4

Records are read sequentially forward or backwards. (Relative file, Indexed file)

```
READ file-name-1 [
{ NEXT     } RECORD]
{ PREVIOUS }
  [INTO identifier-1]
  [WITH [NO] LOCK]
  [AT END imperative-statement-1]
  [NOT AT END imperative-statement-2]
  [END-READ]
```

Syntax Rules

Rules Common to All File Types

1. The storage area of identifier-1 cannot be the same as the storage area of the record area of file-name-1.
2. If the USE AFTER STANDARD EXCEPTION procedure related to file-name-1 is defined, the AT END or INVALID KEY phrases must be used.

F In this implementation of COBOL, however, the USE AFTER STANDARD EXCEPTION procedure and the AT END phrase or INVALID KEY phrase, or both phrases, can be omitted.

F 3. The data item defined in the literal section must not be specified as identifier-1.

4. If identifier-1 is a strongly typed group item, only one group item that is strongly typed with the same type can be specified in the record description entry of file-name-1.

Rules for Relative Files

1. Format 1 reads records from a file using the sequential access method.
2. Format 1 reads records from a file sequentially using the dynamic access method. The NEXT phrase must be specified.
3. Format 2 reads records from a file using the random access method or randomly using the dynamic access method.

Rules for Indexed Files

1. Format 1 reads records from a file using the sequential access method.
2. Format 1 reads records from a file sequentially using the dynamic access method. The NEXT phrase must be specified.
3. Format 3 reads records from a file randomly using the random access method or the dynamic access method.
4. Format 4 reads records sequentially forward or backwards from a file using the dynamic access mode.
5. Data-name-1 must be the name specified in the RECORD KEY or ALTERNATE RECORD KEY phrase of file-name-1. If two or more data-names have been specified in the RECORD KEY phrase when data-name of the RECORD KEY phrase is specified, the row of data-name-1 must be the same as the row of data-name specified in the RECORD KEY phrase, both in the specified sequence and number. If two or more data-names have been specified in the ALTERNATE RECORD KEY phrase when data-name in the ALTERNATE RECORD KEY phrase is specified, the row of data-name-1 must be the same as the row of data-name specified in the ALTERNATE RECORD KEY phrase, both in the specified sequence and number.
6. Data-name-1 can be qualified.

General Rules

Rules Common to Format 1 to Format 3

- Rules Common to ALL file TYPES

1. The file assigned to file-name-1 must be opened in input mode or I-O mode before executing the READ statement.
2. When the READ statement has been executed, the records to be read are selected and the value of the file position indicator of file-name-1 is updated. The value of the I-O status of file-name-1 is also updated. The next operation for the file is determined based on the value of the file position indicator.
3. If the READ statement execution failed, the contents of the record area of file-name-1 are undefined. The file position indicator is updated to indicate that the record area does not contain a valid record.
4. When the value of the file position indicator indicates that the record area of file-name-1 contains a valid record, the record becomes usable. When the INTO phrase is specified, the record is transferred from the record area to identifier-1.
5. When the number of character positions of the record that has been read is less than the minimum record size specified in the record description entry of file-name-1, the contents of the record area that exceed the minimum number of record character positions is undefined. When the number of character positions of the record that has

been read is greater than the minimum record size specified in the record description entry of file-name-1, the record is truncated on the right in accordance with the maximum record size and moved to the record area of file-name-1.

6. The END-READ phrase terminates the range of the READ statement.

- Rules for THE INTO phrase

1. The INTO phrase can be specified in the following cases:
   a. When only one record entry is defined in file-name-1.
   b. When all record-names and identifier-1 related to file-name-1 are group items or alphanumeric data items.
2. When the INTO phrase is specified, it is processed as follows:
   a. The same READ statement without the INTO phrase is executed.
   b. When the READ statement has been executed successfully in (a), the current record is transferred from the record area of file-name-1 to identifier-1 following the rules of a MOVE statement without the CORRESPONDING phrase. The record that has been read can be referenced in both the record area of file-name-1 and the data item of identifier-1.

   The current record size is determined by the size of the record as defined by the RECORD phrase of file-name-1. When the RECORD phrase is specified in the file description entry of file-name-1 in a format other than format 3, the group items are transferred using rule (b) above. If a subscript is specified, identifier-1 is evaluated before processing rule (b), the subscript is evaluated after processing rule (a). If the processing of rule (a) fails, the implicit MOVE statement is not executed.

F

- Rules for record locks

1. When the following READ statements are executed for a file that has specified the LOCK MODE IS AUTOMATIC phrase, the record that is read is locked.
   a. A READ statement with LOCK phrase
   b. A READ statement without LOCK or NO LOCK phrase
2. When a READ statement with the LOCK phrase has been executed for a file that specifies the LOCK MODE IS MANUAL phrase (relative file or indexed file), the record that is read is locked.
3. When a READ statement has been executed for a file opened in input mode, the record is not locked, regardless of whether the READ statement specifies the LOCK phrase or not.
4. If a record to be read by the READ statement is locked (either implicitly or explicitly) by another file connector, the READ statement fails. A value indicating that the record is locked is moved to the I-O status of the file and the value of the file position indicator is not changed.

Rules for Format 1

- Rules Common to ALL file TYPES

1. When a sequential file, a relative file using the sequential access method, or an indexed file using the sequential access method is specified as file-name-1, the NEXT phrase may be omitted.
2. If file-name-1 uses the sequential access method, a relative file or an indexed file using the sequential access method, the READ statement reads the next record from the file assigned to file-name-1.
3. Relative and indexed files using the dynamic access method that execute a READ statement with the NEXT phrase for file-name-1 will read the next record from the file assigned to file-name-1.
4. When the READ statement has been executed, the value of the file position indicator is determined in accordance with the rules for determining a usable record (see below). The next operation of the READ statement is determined by the value of the file position indicator.

- Rules for determining a usable record (sequential files)

    1. The value of the file position indicator at the start of the execution of the READ statement is used to determine the usable record in accordance with the following rules. Records in sequential files are selected by their record numbers.

        a. When the file position indicator indicates that there are no remaining valid records, the READ statement fails.

        b. When the file position indicator indicates that the file is not an optional file, a file at end condition occurs.

        c. When the file position indicator indicates that an OPEN statement was previously executed, the first record in the file that has a record number greater than or equal to the file position indicator is read.

        d. When the file position indicator indicates that a READ statement was previously executed, the first record in the file that has a record number greater than the file position indicator is read.

    2. If no record satisfies conditions (c) and (d) above, an "at end" condition occurs. The file position indicator is updated to indicate that there are no more records to be read.

    3. If a record that satisfies conditions (c) and (d) above is found, the record becomes usable in the record area related to file-name-1 and the file position indicator is updated with the record number of the usable record.

- Rules for determining a usable record (relative files)

    1. The value of the file position indicator when a READ statement is executed is used to determine the usable record in accordance with the following rules. Records in relative files are selected by their relative key numbers.

        a. If the file position indicator indicates that there are no remaining valid records, execution of the READ statement fails.

        b. When the file position indicator indicates that the file is an optional file, an "at end" condition occurs.

        c. If the file position indicator indicates that an OPEN statement was previously executed, the first record in the file that has a record number greater than or equal to the file position indicator is read.

        d. If the file position indicator indicates that a START statement was previously executed, a record having a relative record number equal to the file position indicator is read.

        e. If the file position indicator indicates that a READ statement was previously executed, the first record that has a relative record number greater than the file position indicator is read.

    2. If no record satisfies conditions (c) and (d) above, a file at end condition occurs. The file position indicator is updated to indicate that there are no more records to be read in the file.

    3. If a record that satisfies conditions (c) and (d) above is found, processing will be as follows:

        a. If file-name-1 specifies the RELATIVE KEY phrase, the number of valid digits of the relative record number in the selected record and the size of the relative key item are compared.

            - If the number of valid digits of the relative record number is greater than the size of the relative key item, an "at end" condition occurs, and the I-O status is updated with a value to indicate that the number of valid digits of the relative record number exceeded the size of the relative key.

            - If the number of valid digits of the relative record number is less than the size of the relative key item, the selected record becomes usable in the record area in file-name-1 and the relative record number of the usable record is moved to the file position indicator. The relative record number of the usable record is transferred to the relative key item in accordance with the rules for the movement of data.

        b. If file-name-1 does not specify the RELATIVE KEY phrase, the selected record becomes usable in the record area of file-name-1, and the relative record number of the usable record is moved to the file position indicator.

- Rules for determining a usable record (indexed file)

1. The value of the file position indicator when a READ statement is executed is used to determine the usable record in accordance with the following rules. Records in an indexed file are selected by values of the keys referenced.

   a. If the file position indicator indicates that there are no remaining valid records, execution of the READ statement fails.

   b. If the file position indicator indicates that the file is an optional file, an "at end" condition occurs.

   c. If the file position indicator indicates that an OPEN statement was previously executed, the first record in the file that matches the current key referenced is read.

   d. If the file position indicator indicates that a START statement was previously executed, the record key value specified in the file position indicator is read.

   e. If the file position indicator indicates that a READ statement was previously executed, a record that satisfies either of the following conditions is read:

      - A record that is logically allocated just after the record that became usable by execution of the previous READ statement.

F

      - If the direction for searching is reversed by execution of a START statement with the REVERSED ORDER phrase, the record which logically precedes the record that became usable by a previous READ statement is selected.

2. If no record satisfying conditions (c), (d), and (e) above is found, an "at end" condition occurs. The file position indicator is updated to indicate that no remaining records can be read.

3. If a record satisfying the conditions (c), (d), and (e) above is found, the record becomes usable in the record area of file-name-1. At the same time, the current key of reference of the usable record is moved to the file position indicator.

4. If execution of the READ statement fails, the value of the key of reference is undefined.

5. When a READ statement has been executed with the prime record key(*) or alternate record key as the key of reference, there may be multiple records with the same key values (duplicate keys). In that case, records are read in the following sequence:

F

   a. When the file position indicator is set by a START statement with the REVERSED ORDER phrase, records are read sequentially in reverse order of their keys.

   b. Otherwise, records are read sequentially in the order of their keys.

6. If a format 3 READ statement is executed followed by a format 1 READ statement and the file is using the dynamic access method, the format 1 READ statement uses the key of reference as follows:

   a. The key of reference used by the format 3 READ statement is used until the key of reference is changed.

F

   b. Records are read in the normal ascending order unless a START statement with the REVERSED ORDER phrase is executed.

F

* : Extended functions or functions specific to NetCOBOL.

- prime record key

- Transfer OF Control WHEN EXCEPTIONS OCCUR

1. When an "at end" condition occurs during execution of a READ statement, the READ statement execution fails. A value indicating the at end condition occurred is moved to the I-O status of file-name-1, the control is transferred according to the rules shown in the topic titled "AT END Phrase" in the "Common Statement Rules" section earlier in this chapter.

2. When an exception condition other than the at end condition occurs during execution of the READ statement, the file position indicator and the I-O status of file-name-1 are updated and control is transferred according to the rules shown in the topic titled "AT END phrase," in the "Common Statement Rules" section earlier in this chapter.

3. If no exception condition occurs during execution of a READ statement, processing will be as follows:

    a. The file position indicator and the I-O status of file-name-1 are updated.

    b. The record that has been read becomes usable in the record area.

    c. Control is transferred according to the rules shown in the topic titled "AT END phrase" in the "Common Statement Rules" section earlier in this chapter.

- Rules related to reel/unit file AND sequential files

  When the end of a reel/unit during execution of a READ statement for a sequential file has been detected, or when no record exists in the reel/unit, the following processing is performed if it is not the logical end of the file.

  a. The standard procedure for end reel/unit labels is performed.

  b. The reel/unit is replaced. The volume indicator is updated to indicate the next reel/unit.

  c. The standard procedure for start reel/unit labels is performed.

F

- Rules for line sequential files

  When a READ statement has been executed for a line sequential file, record data will be updated as follows. In the explanation below, " the length of the data read, up to the record delimiter" does not include the delimiter for the record.:

  a. When the length of the data read, up to the record delimiter, is equal to the maximum record length.

  b. When the length of the data read, up to the record delimiter, is shorter than the maximum record length, the data up to the record delimiter is moved to the record area and the remaining characters of the record area are space filled.

  c. When the length of the data read, up to the record delimiter, is longer than the maximum record length, the entire record data is read by executing two or more READ statements. Execution of the first READ statement transfers data, beginning with the first character, to the record area, up to the maximum record length of the record. Execution of the second and subsequent READ statements transfers data beginning with the next character position of the data that was read by the preceding READ statement. Rules (a) to (c) are applied in repetitive execution of READ statements.

Rules for Format 2

1. For input files, when the file position indicator indicates that the file is not an optional file, and the file does not exist, execution of a READ statement will cause an invalid key condition and the READ statement fails.

2. The relative record number of the record to be read must be moved to the relative key item of file-name-1 before execution of the READ statement. When a READ statement is successfully executed, the value of the relative key item read is moved to the file position indicator. If a record having a relative record number equal to the value of the file position indicator is found, the record becomes usable in the record area related to file-name-1, otherwise an "at end" condition occurs and execution of the READ statement fails.

Rules for Format 3

1. For input files, when the file position indicator indicates that the file is not an optional file, and the file does not exist, execution of a READ statement will cause an invalid key condition and the READ statement fails.

2. When the KEY phrase is specified, data-name-1 becomes the key of reference for the READ statement.

3. When the KEY phrase has been omitted, the prime record key becomes the key of reference for the READ statement.

4. The value of the key of reference is moved to the file position indicator by execution of the READ statement. To select the record to be read from the file, the data corresponding to the record key of reference is compared with the value of the file position indicator. The comparison operation repeats until the first record in which the data corresponding to the record key of reference matches with the value of the file position indicator is found.

   When a record satisfying the condition is found as a result of the comparison, the record becomes usable in the record area related to file-name-1. If no record having a relative record number equal to the value of the file position indicator is found, an "at end" condition occurs and execution of the READ statement fails.

5. If execution of a READ statement fails, the value of the key of reference is undefined.

- Rules Common to Format 2 and Format 3

1. When an invalid key condition occurs during execution of a READ statement, a value indicating such is moved to the I-O status of file-name-1 and control is transferred according to the rule shown in the topic titled "INVALID KEY Phrase" in the "Common Statement Rules" section earlier in this chapter.
2. If an exceptional condition other than the invalid key condition occurs during execution of a READ statement, the file position indicator and the I-O status of file-name-1 is updated and control is transferred according to the rules shown in the topic titled "INVALID KEY Phrase" in the "Common Statement Rules" section earlier in this chapter.
3. When execution of a READ statement is successful, the following processing is performed:
    a. The file position indicator and the I-O status of file-name-1 are updated.
    b. The record that has been read becomes usable in the record area. The implicit transfer is performed if the INTO phrase is specified.

Control is transferred according to the rule described in the topic titled "INVALID KEY Phrase" in the "Common Statement Rules" section earlier in this chapter.

Rules for Format 4

1. Format 4 READ statements specifying the NEXT phrase retrieves the next logical record in a forward direction from file-name-1. The next logical record is determined by the following rules:
    a. If the file position indicator was positioned by execution of an OPEN statement, the record pointed to by the file position indicator is made available.
    b. If the file position indicator was positioned by execution of a previous READ or START statement, thus establishing a key of reference, the file position indicator is updated to point to the next record in the key of reference and the record is made available.

       However, if a READ statement indicates that the record to read next is locked, the file position indicator is updated to point to the locked record. Subsequent format 4 READ statements will retrieve the same record.
2. If a READ statement is executed, and no logical record exists for the key of reference, an "at end" condition occurs.
3. Format 4 READ statements specifying the PREVIOUS phrase retrieves the previous logical record in a backward direction from file-name-1. The next logical record is determined by the following rules:
    a. If the file position indicator was positioned by execution of an OPEN statement, a READ statement specifying the PREVIOUS clause will cause an "at end" condition, and the READ statement will be unsuccessful.
    b. If the file position indicator was positioned by execution of a previous READ or START statement, thus establishing a key of reference, the file position indicator is updated to point to the previous record in the key of reference and the record is made available.

       However, if a READ statement indicates that the record to read next is locked, the file position indicator is updated to point to the locked record. Subsequent format 4 READ statements will retrieve the same record.

F

6.4.35 READ Statement (Presentation File)

The READ statement reads a record from a file.

Format

READ file-name-1 [NEXT] RECORD

[INTO identifier-1]

[AT END imperative-statement-1]

[NOT AT END imperative-statement-2]

[END-READ]

Syntax Rules

1. The storage area of identifier-1 must not be the same as the storage area of the record area of file-name-1.
2. When a USE AFTER STANDARD EXCEPTION procedure related to file-name-1 is not defined, the AT END or INVALID KEY must be specified. In this implementation of COBOL, however, both the USE AFTER STANDARD EXCEPTION procedure and the AT END phrase can be omitted.
3. A data item defined in the CONSTANT SECTION must not be specified as identifier-1.

General Rules

1. The file assigned to file-name-1 must be opened in the input or I-O mode before execution of a READ statement.
2. When a READ statement is executed, the value of the I-O status of file-name-1 is updated.
3. The NEXT phrase can be omitted. The NEXT phrase has no affect on execution of a READ statement.
4. When execution of a READ statement begins, the file is checked for existence of the next valid record. If one exists, the record becomes usable in the record area related to file-name-1. If there is no valid record, an "at end" condition occurs.
5. The INTO phrase can be specified under the following conditions:
    a. When only one record description entry of file-name-1 is defined.
    b. When all record-names and identifier-1 related to file-name-1 are group items or alphanumeric data items.
6. When the INTO phrase is specified, processing will be as follows:
    a. The READ statement without the INTO phrase is executed.
    b. When the READ statement has been executed successfully in (a), the current record is transferred from the record area of file-name-1 to identifier-1 according to the MOVE statement rule without the CORRESPONDING phrase. The record that was read becomes usable in both the record area of file-name-1 and the data item of identifier-1. The current record size is determined in accordance with the rules of the RECORD phrase of file-name-1. When the RECORD VARYING phrase is specified in the file description entry of file-name-1, the group items are transferred when processing (b). If a subscript is specified for identifier-1, it is evaluated before processing (b). If process (a) fails, the implicit MOVE statement is not executed.
7. When an "at end" condition occurs during execution of a READ statement, the READ statement execution fails. After the value indicating the at end condition occurred is moved to the I-O status of file-name-1, the control is transferred according to the rules described in the topic titled "AT END Phrase" in the "Common Statement Rules" section earlier in this chapter.
8. When an exceptional condition other than an "at end" condition occurs during execution of a READ statement, the file position indicator and the I-O status of file-name-1 are updated, and control is transferred according to the rules described in the topic titled "AT END Phrase" of the "Common Statement Rules" section earlier in this chapter.
9. If the READ statement is successful, processing will be as follows:
    a. The file position indicator and the I-O status of file-name-1 are updated.
    b. The record that has been read becomes usable in the record area.
    c. The control is transferred according to the rules described in the topic titled "AT END Phrase" in the "Common Statement Rules" section earlier in this chapter.
10. If execution of the READ statement fails, the contents of the record area of file-name-1 are undefined.
11. When the number of record character positions of a record that is read is less than the minimum record size specified in the record description entry of file-name-1, the contents of file-name-1's record area that exceed the number of record character positions will be undefined. The number of record character positions of a read record may be more than the maximum record size specified in the record description entry of file-name-1. In this case, the right side of the record that is read is truncated to the maximum record size. The READ statement is executed successfully in either case. A value indicating a record length conflict is moved in the I-O status of file-name-1.

12. The END-READ phrase terminates the range of a READ statement.

6.4.36 RELEASE Statement (Sort-merge)

The RELEASE statement delivers a record to the first step of a sorting operation.

Format

RELEASE record-name-1 [FROM identifier-1]

Syntax Rules

1. The RELEASE statement can only appear in the input procedure specified by the SORT statement for the file assigned to record-name-1.
2. Record-name-1 must be defined in a record description entry related to the file description entry for the sort-merge file.
3. Record-name-1 can be qualified by file-name.
4. When identifier-1 specifies a data item, the storage area of record-name-1 must not be the same as that of identifier-1. When identifier specifies a function-identifier, identifier-1 must be an alphanumeric function.

General Rules

1. The RELEASE statement delivers a record assigned to record-name-1 to the first step of a sorting operation.
2. When the RELEASE statement has been executed, the record assigned to record-name-1 becomes usable in the record area. However, when the file related to record-name-1 is specified in the SAME RECORD AREA clause in the I-O-CONTROL paragraph, the record assigned to record-name-1 will not be usable in the record area. The record is usable as an output record of another file specified in the SAME RECORD AREA clause.
3. Executing the RELEASE statement with the FROM phrase produces the same results as executing the following two statements in the following sequence:
    a. MOVE identifier-1 TO record-name-1
    b. The same RELEASE statement without the FROM phrase

6.4.37 RETURN Statement (Sort-merge)

The RETURN statement receives the sorted or merged record from the last step of a sort or merge operation.

Format

RETURN file-name-1 RECORD [INTO identifier-1]

AT END imperative-statement-1

[NOT AT END imperative-statement-2]

[END-RETURN]

Syntax Rules

1. The RETURN statement can only appear in the output procedure specified by the SORT or MERGE statement for the file assigned to file-name-1.
2. File-name-1 must be defined in the file description (SD) entry of the sort-merge file.
3. The storage area of identifier-1 must not be the same as that of file-name-1.
4. F When identifier-1 is a strongly typed group item, only one group item that is strongly typed with the same type can be specified in the record description entry of file-name-1.

General Rules

1. When the RETURN statement has been executed, the next record determined by the row of keys specified in the SORT or MERGE statement is retrieved from the file assigned to file-name-1. If there is no record to retrieve, an "at end" condition occurs.
2. Records defined in the record description entry of file-name-1 share the same storage area in accordance with redefinition rules.
3. 3. The INTO phrase can be specified when the following conditions are met:
    a. Only one record description entry of file-name-1 is defined.
    b. All record-names and identifier-1 related to file-name-1 are group items or alphanumeric data items.
4. When the INTO phrase is specified, the processing is as follows:
    a. The same RETURN statement without the INTO phrase is executed.
    b. When the RETURN statement has been executed successfully in (a), the current record is transferred from the record area of file-name-1 to identifier-1 accordance with MOVE statement rules without the CORRESPONDING phrase. When the RECORD VARYING phrase is specified in the file description entry of file-name-1, group items are moved when processing (b). If identifier-1 is subscripted, the subscript is evaluated before processing (b). If the processing of (a) fails, the implicit MOVE statement is not executed.
5. When an "at end" condition occurs, execution of the RETURN statement fails. After control has been transferred to imperative-statement-1 and that statement has been executed, control transfers to the end of the RETURN statement. But, when a procedure branching statement is executed in the imperative statement or the conditional statement causing an explicit transfer of control, control transfers according to the statement rule. After execution of imperative-statement-1, the RETURN statement cannot be executed as a part of the output procedure during execution.
6. If the RETURN statement is executed successfully, processing will be as follows:
    a. The record that was input becomes usable in the record area of file-name-1. When the INTO phrase is specified, the record is transferred from the record area of file-name-1 to the record area of identifier-1.
    b. If the NOT AT END phrase is specified, control transfers to imperative-statement-2. After the execution of imperative-statement-2, control transfers to the end of the RETURN statement. However, if imperative-statement-2 or the conditional statement executes a procedure branching statement causing an explicit transfer of control, control is transferred according to the statement rules. When the NOT AT END phrase has been omitted, control is transferred to the end of the RETURN statement.
7. The END-RETURN phrase terminates the range of the RETURN statement.

6.4.38 REWRITE Statement (Sequential File, Relative File, Indexed File)

The REWRITE statement logically rewrites a record of a mass storage file.

Format 1

Records are rewritten in the order that they are read. (Sequential file, relative file)

```
REWRITE record-name-1 [FROM identifier-1]
    [END-REWRITE]
```

Format 2

The specified record is rewritten. (Relative file, indexed file)

```
REWRITE record-name-1
    [FROM identifier-1]
    [INVALID KEY imperative-statement-1]
```

[NOT INVALID KEY imperative-statement-2]

[END-REWRITE]

Syntax Rules

Rules Common to All Files

1. Record-name-1 must be defined in the record description entry related to the file description entry associated file-name. Record-name-1 can be qualified by file-name.
2. When identifier-1 specifies a data item, the storage area of record-name-1 must not be the same as that of identifier-1. When identifier-1 specifies a function identifier-1, identifier-1 must be an alphanumeric function.

Rules for Relative Files

1. If the file of record-name-1 is in sequential access mode, a format 1 REWRITE statement must be used.
2. If the file of record-name-1 is in random or dynamic access mode, a format 2 REWRITE statement must be used. If the USE AFTER STANDARD EXCEPTION procedure related to record-name-1 is not defined, the INVALID KEY must be specified.

F

In this implementation of COBOL, however, both the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.

Rules for Indexed Files

1. The file related to record-name-1 must not be a file in the random access mode in which the RECORD KEY phrase with the DUPLICATES phrase is specified.
2. When the USE AFTER STANDARD EXCEPTION procedure related to record-name-1 is not defined, the INVALID KEY phrase must be specified.

F

In this implementation of COBOL, however, both the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.

General Rules

Rules Common to All Files

1. The file related to record-name-1 must be a mass storage file.
2. The file related to record-name-1 must be opened in I-O mode before executing the REWRITE statement.
3. When the REWRITE statement has been executed successfully, the record assigned to record-name-1 remains usable in the record area. However, when the file related to record-name-1 is specified in the SAME RECORD AREA phrase in the I-O-CONTROL paragraph, the record assigned to record-name-1 will not be usable by the REWRITE statement. Other statements in the program can continue to refer to the record.
4. The result of executing the REWRITE statement with FROM phrase is the same as the result of executing the following two statements in sequence:
    a. MOVE identifier-1 TO record-name-1
    b. The same REWRITE statement without FROM phrase
5. After the REWRITE statement has been executed, the information in record-name-1 area cannot be used except in cases where the file related to record-name-1 is specified in the SAME RECORD AREA clause. However, information in identifier-1 area is usable.
6. Execution of the REWRITE statement does not change file position indicator.
7. When the REWRITE statement is executed, the value of the I-O status of the file related to record-name-1 is updated.
8. The record is delivered to the I-O-CONTROL system by executing the REWRITE statement.

F

9. If an attempt is made to rewrite a record locked by another file connector, execution of the REWRITE statement fails, and a value indicating that the record is locked is moved to the I-O status related to record-name-1.
10. If AUTOMATIC is specified in the LOCK MODE phrase of the file related to record-name-1, the existing record is released from the lock if the REWRITE statement has been executed successfully.
11. The END-REWRITE phrase terminates the range of the REWRITE statement.

Rules for Sequential Files

1. The READ statement must be executed for the file related to record-name-1 prior to executing the REWRITE statement. If a REWRITE statement is executed after a READ statement has been executed successfully, the record read by the READ statement is logically rewritten.
2. If the length of the record assigned to record-name-1 differs from the length of the record to be rewritten, execution of the REWRITE statement fails and no record will be rewritten. The contents of the record area of record-name-1 are unchanged and the following processing will be performed:
    a. The value indicating that the record lengths are not the same is moved to the I-O status of the file related to record-name-1.
    b. If the USE AFTER STANDARD EXCEPTION procedure of the file related to record-name-1 has been defined, the procedure is executed. After the execution of the procedure, control is transferred according to the rules for the USE statement. If the FILE STATUS phrase is specified in the file related to record-name-1 and the USE AFTER STANDARD EXCEPTION procedure has not been defined, control is transferred to the end of the REWRITE statement. If neither the USE AFTER STANDARD EXCEPTION procedure is defined nor the FILE STATUS clause is specified, the processing results are undefined.

Rules for Relative Files

1. If the file to be rewritten is in sequential access mode, a format 1 REWRITE statement must be preceded by a READ statement of the file related to record-name-1. If the READ statement has been executed successfully, the record that has been read by the READ statement is rewritten by executing the REWRITE statement.
2. If the file to be rewritten is in random or dynamic access mode, a format 2 REWRITE statement will rewrite the record containing the same relative record number as the value of the relative key item. If no record in the file has the same relative record number as the value of the relative key item, an invalid key condition occurs.

Rules for Indexed Files

1. Under the following conditions, a READ statement for the file related to record-name-1 must be executed prior to execution of a REWRITE statement. If the REWRITE statement is executed after execution of a successful READ statement, the record read by the READ statement is rewritten.
    a. A REWRITE statement for a file in sequential access mode
F
    b. A REWRITE statement for a file in which the RECORD KEY clause with the DUPLICATES phrase is specified
    c. A REWRITE statement for a file in which the file position indicator has been set by a START statement with the REVERSED ORDER phrase
2. In the REWRITE statement for the file in sequential access method, the record to be rewritten is specified by the value of the prime record key. To execute the REWRITE statement, the value of the prime record key must be equal to the value of the prime record key of the record which has been read last from this file.

F

3. When executing a REWRITE statement for a file in which the RECORD KEY clause with the DUPLICATES phrase is specified or a file in which the file position indicator has been set by a START statement with the REVERSED ORDER phrase, the record to be rewritten is specified by the value of the prime record key. To execute the REWRITE statement, the prime record key must be equal to the prime record key of the last record read from the file.
4. For files in random access or dynamic access mode, the record to be rewritten is specified by the value of the prime record key.

5. When the ALTERNATE RECORD KEY clause is specified for the file related to record-name-1, the REWRITE statement is executed depending on the value of the alternate record key as follows:

    a. If the value of the alternate key does not change, the search order of the record does not change even though the key is a key of reference.

    b. If the value of the alternate key changes, the search order of the record may change if the key is a key of reference. When multiple key values are allowed, the record is logically located at the end of the group of records containing the same alternate record key as the specified alternate record key.

6. 6. An invalid key condition occurs in the following cases:

    a. In REWRITE statements that satisfy the following conditions, when the value of the prime record key in the record to be rewritten is not equal to the value of the prime record key of the record that was last read from the same file, the invalid key condition occurs.

    - A REWRITE statement of a file in sequential access mode

F

    - A REWRITE statement of a file in which the RECORD KEY clause with the DUPLICATES phrase is specified

    - A REWRITE statement of a file in which the file position indicator has been set by a START statement with the REVERSED ORDER phrase

    b. For files in the dynamic or random access mode, if a REWRITE statement is executed and no record in the file has the same prime record key as the record to be rewritten, an invalid key condition occurs.

    c. For files that specify the ALTERNATE RECORD KEY clause without the DUPLICATES phrase, if no record in the file has the same alternate record key as the record to be written, execution of the REWRITE statement will result in an invalid key condition.

Rule Common to Relative and Indexed Files

1. The number of character positions of the record assigned to record-1 need not be the same as the length of the record to be rewritten.

2. The number of character positions of the record assigned to record-1 must not be greater than the maximum record length of the file related to record-name-1. In that case, the REWRITE statement execution fails and the record is not rewritten. The contents of the record area in the file related to record-name-1 are not changed. A value indicating this situation is moved to the I-O status of the file related to record-name-1.

3. When an invalid key condition occurs during execution of the REWRITE statement, the REWRITE statement execution fails and the record is not rewritten. A value indicating that an invalid key condition has occurred is moved to the I-O status of record-name-1 and control is transferred according to the rules described in the topic titled "INVALID KEY Phrase" in the "Common Statement Rules" section earlier in this chapter.

4. If an exception condition other than an invalid key condition occurs during execution of the REWRITE statement, the file position indicator and I-O status of record-name-1 are updated and control is transferred according to the rules described in the topic titled "INVALID KEY Phrase" of the "Common Statement Rules" section earlier in this chapter.

5. When execution of the REWRITE statement is successful, the following processing will be performed:

    a. The I-O status of the file related to record-name-1 is updated.

    b. The record that has been read by the READ statement is rewritten.

    c. Control is transferred according to the rules described in the topic titled "INVALID KEY Phrase" in the "Common Statement Rules" section earlier in this chapter.

6.4.39 SEARCH Statement (Nucleus)

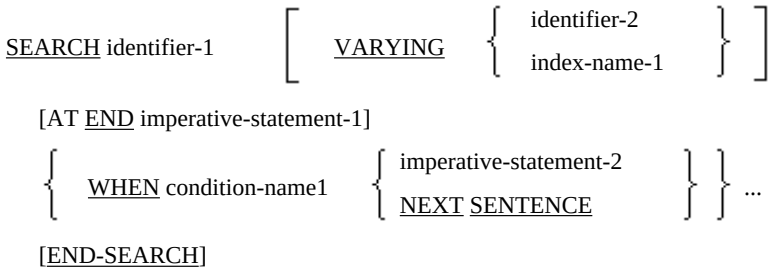
The SEARCH statement searches table elements.

For details about how to use the examples, refer to "SEARCH Statement" in the "Syntax Samples".

Format 1

Table elements are searched in sequence.

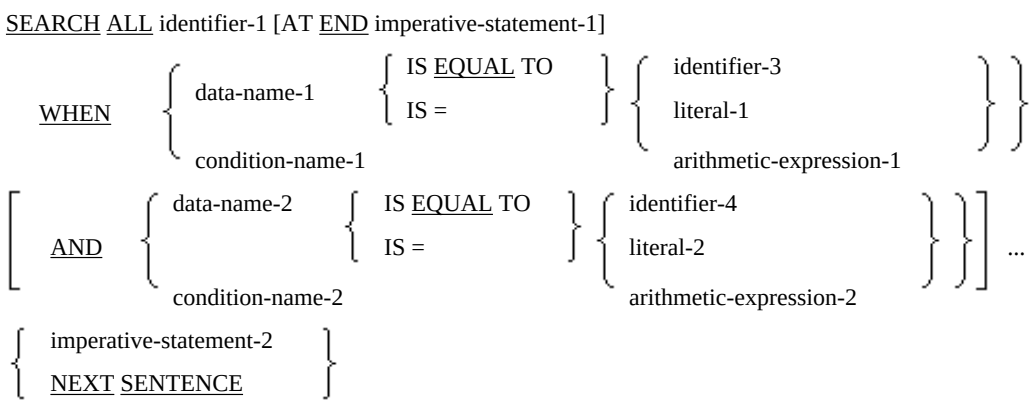
```
SEARCH identifier-1 [ VARYING { identifier-2 } ]
                              { index-name-1 }
  [AT END imperative-statement-1]
  { WHEN condition-name1 { imperative-statement-2 } } ...
                         { NEXT SENTENCE          }
  [END-SEARCH]
```



Format 2

Table elements aligned in ascending or descending order are searched.

```
SEARCH ALL identifier-1 [AT END imperative-statement-1]
  WHEN { data-name-1 { IS EQUAL TO } { identifier-3 } }
       { condition-name-1 { IS =  } { literal-1 } }
                                    { arithmetic-expression-1 }
[ AND { data-name-2 { IS EQUAL TO } { identifier-4 } } ] ...
      { condition-name-2 { IS =  } { literal-2 } }
                                   { arithmetic-expression-2 }
{ imperative-statement-2 }
{ NEXT SENTENCE          }
  [END-SEARCH]
```



Note

The symbol = is required, but not underlined to avoid confusion with other symbols.

Syntax Rules

Rules for Format 1

1. Identifier-1 cannot be subscripted or reference modified. Identifier-1 must be a data item which specifies the OCCURS clause with the INDEXED BY phrase.
2. Identifier-2 must be an index data item or integer item. The first or only index-name among index-names written in items in which the OCCURS clause with INDEXED BY phrase is specified in identifier-1 must not be used for adding the subscript to identifier-2.

Rules for Format 2

1. Identifier-1 must be a data item which specifies the OCCURS clause with the INDEXED BY phrase and the KEY IS phrase. identifier-1 cannot be subscripted or reference modified.
2. Data-name-1 and data-name-2 must be data-names specifying the KEY IS phrase of the OCCURS clause of identifier-1.
3. Conditional variables for condition-name-1 and condition-name-2 must be data-names specifying in the KEY IS phrase of the OCCURS clause of identifier-1. The THRU phrase must not be specified in the VALUE clause of the data description entry in condition-name-1 and condition-name-2.

4. To subscript data-name-1, data-name-2, condition-name-1, or condition-name-2, the subscript must be the first index-name in the INDEXED BY phrase of the OCCURS clause of identifier-1.

5. The data-name specified in the KEY IS phrase in the OCCURS clause of identifier-1 cannot be specified as identifier-3, identifier-4, the identifier in arithmetic expression-1, or the identifier in arithmetic expression-2. To add a subscript to either of these identifiers, the subscript must not be the first index-name in the INDEXED BY phrase in the OCCURS clause of identifier-1.

6. When two or more data-names have been specified in the KEY IS phrase of the OCCURS clause in identifier-1, all data-names in the KEY IS phrase and all conditions related to those data-names must be specified in a WHEN phrase. Data-name and condition-name can be specified in one WHEN phrase.

F 7. Data-name-1, data-name-2, identifier-3, identifier-4, literal-1 and literal-2 cannot be Boolean data items.

Rules Common to All Formats

The NEXT SENTENCE and the END-SEARCH phrase are mutually exclusive.

Example

Example of a Format 2 SEARCH Statement

When the following table is defined in the DATA DIVISION, SEARCH statements (a) and (b) can be written in the PROCEDURE DIVISION.

```
01  TBL.
    02  Y OCCURS 50 TIMES
          ASCENDING KEY IS K1, K2, K3, K4
          INDEXED BY  IX1.
        03  K1 PIC 9.
            88  T1 VALUE 3.
        03  K2 PIC 9.
            88  T2 VALUE 2.
        03  K3 PIC 9.
            88  T3 VALUE 1.
        03  K4 PIC 9.
77  X PIC 9.
```

(a)

```
    SEARCH ALL Y WHEN K1(IX1) = 3
                  AND K2(IX1) = 2
                  AND K3(IX1) = 1
    *>  :
    END-SEARCH
```

(b)

```
    SEARCH ALL Y WHEN T1(IX1)
                  AND T2(IX1)
                  AND T3(IX1)
                 AND K4(IX1) = X
    *> :
    END-SEARCH
```

General Rules

Rules for Format 1

1. A format 1 SEARCH statement searches table elements in the identifier-1 table in sequence. These elements are indicated by index-names for table lookup. When a table element satisfying condition-1 is found, the processing of imperative-statement-2 corresponding to condition-1 is performed.

2. To search table elements, the following index-names are used. These index-names are called "index-names for table lookup."

    a. The first index-name specified in the INDEXED BY phrase of the OCCURS clause of identifier-1 is used in the following cases:

        - When the VARYING phrase has been omitted.

        - When the index-name in the table that is not identifier-1 table specifies the VARYING phrase.

        - When identifier-2 specifies the VARYING phrase.

    b. When the first index-name in the INDEXED BY phrase of the OCCURS clause of identifier-1 is specified, index-name is used.

3. Table elements are searched follows:

    a. When the value of the index-name used for table lookup is greater than the value corresponding to the maximum occurrence number of identifier-1, table elements are not searched. In that case, control transfers to imperative-statement-1 if the AT END phrase is specified. If the AT END phrase has been omitted, control transfers to the end of the SEARCH statement.

    b. When the value of the index-name used for table lookup is less than the value corresponding to the maximum occurrence number of identifier-1, processing is as follows:

        (1) The table element indicated by the value of the index-name used for table lookup is tested to see whether it satisfies condition-1. The test is performed in the order of the elements in which the WHEN phrase was written.

        (2) If no condition has been satisfied, the value of the index-name used for lookup is increased. Then process (1) repeats.

        (3) When the new value of the index-name used for lookup is not within the range of occurrence numbers of the table, the SEARCH statement processing terminates in the same way as the processing described in (a).

        (4) When one of the conditions is satisfied, the table element test terminates. If imperative-statement-2 is specified, control transfers to imperative-statement-2. If the NEXT SENTENCE is specified, control transfers to the next executable sentence. At that time, the value of the index-name used for lookup corresponds to the occurrence number of the table element that satisfied the condition.

4. The values of the index-name used for lookup, identifier-2 and index-name-1, are increased according to the following rules:

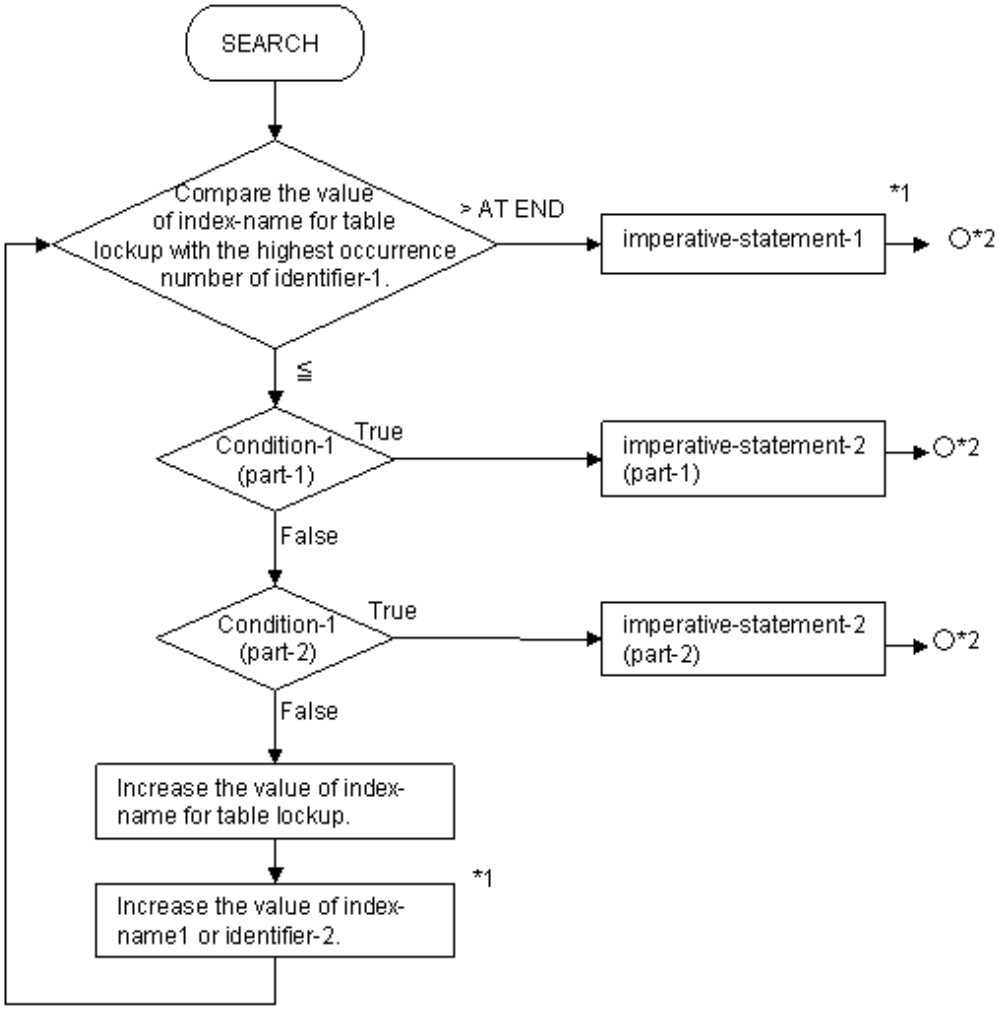
    a. When the VARYING phrase has been omitted, or index-name specifies the INDEXED BY phrase of the OCCURS clause of identifier-1 and identifier-1 is specified in the VARYING phrase of the SEARCH statement, only the index-name for the table being searched is increased.

    b. When an index-name in a table other than identifier-1 table is specified in index-name-1 of the VARYING phrase of the SEARCH statement, the value of index-name-1 increases by the same amount as the index-name for table lookup increases.

    c. When an index data item is specified in the VARYING phrase of the SEARCH statement, the value of the index data item increases by the same amount as the index-name for table lookup increases.

    d. When a literal item is specified in the VARYING phrase of the SEARCH statement, the value of the literal item increases by one.

5. When two WHEN phrases have been written, the table element test is performed according to the following flow chart:



*1 This operation is performed only when the SEARCH statement has been written.

*2 Control transfers to the end of the SEARCH statement after the execution of the imperative statement. However, when a procedure branching statement has been written in the imperative statement that causes an explicit transfer of control, control transfers according to the statement rules.

Rules for Format 2

1. The SEARCH statement searches table elements based on KEY value to satisfy the WHEN phrase condition in identifier-1 table. When a table element that satisfies the condition is found, imperative-statement-2 is executed.
2. The first index-name specified in the INDEXED BY phrase of the OCCURS clause of identifier-1 is used for searching the table elements. Index-name is called an "index-name for table lookup." Only the value of the index-name used for lookup is changed.
3. The execution results of a format 2 SEARCH statement are defined only when all the following conditions are satisfied.
    a. Data in identifier-1 table is aligned in ascending or descending order according to the description of the KEY IS phrase in the OCCURS clause of identifier-1.
    b. Values of data-name-1 and data-name-2, or values of condition-name-1 and condition-name-2 are values that enable table elements to be identified.

4. Regardless of the values of the index-names used for table lookup, table elements are searched according to the following procedure:

   a. Table elements to satisfy all of the conditions specified in the WHEN phrase are searched.

   b. If no table element satisfies all of the conditions specified in the WHEN phrase is found, control transfers to imperative-statement-1 when the AT END phrase has been written. When the AT END phrase has been omitted, control transfers to the end of the SEARCH statement and the value of the index-name for the table is undefined.

   c. If a table element that satisfies all the conditions in the WHEN phrase is found, control transfers to imperative-statement-2 if it exists. If imperative-statement-2 is NEXT SENTENCE, control transfers to the next executable sentence. The value that corresponds to the occurrence number of the table element that satisfied the condition is moved to the index-name of the table.

Rules Common to All Formats

1. The end of the SEARCH statement is specified by any of the following methods:

   a. The END-SEARCH phrase

   b. A separator period

   c. An ELSE phrase corresponding to an IF statement or END-IF phrase when the SEARCH statement is the imperative statement of the IF statement

   See the topic "Scope of Statements" in the "Composition of the Procedure Division" section earlier in this chapter for the range of an IF statement.

2. After execution of imperative-statement-1 or imperative-statement-2, control transfers to the end of the SEARCH statement. However, if the imperative statement contains a procedure branching statement that causes an explicit transfer of control, control is transferred according to the statement rules.

6.4.40 SET Statement (Nucleus)

The SET statement sets a table element index, an external switch status, or a condition variable value.

Format 1

Table element indexes are set.

```
                 { index-name-1   }                  { index-name-2   }
SET              {                } ... TO           { identifier-2   }
                 { identifier-1   }                  { integer-1      }
```

Format 2

The value of index-name is increased or decreased.

```
                                  { UP BY      }  { identifier-3 }
SET {index-name-3} ...            {            }  {            }
                                  { DOWN BY    }  { integer-2    }
```

Format 3

An external switch status is set.

```
                                                  { ON  }  }
SET              { {mnemonic-name-1}... TO ...    {     }  }
                                                  { OFF }  }
```

Format 4

The value of a condition variable is set.

SET {condition-name-1}... TO TRUE

Syntax Rules

1. Identifier-1 and identifier-2 must be index data items or integer items.
2. Identifier-3 must be an integer item.
3. A non-zero positive integer can be specified as integer-1. A non-zero positive integer or negative integer can be specified as integer-2.
4. Mnemonic-name-1 must be related to one of the keywords SWITCH-0 through SWITCH-7 or SWITCH-1 through SWITCH-8 in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

General Rules

Rules for Format 1

1. A format 1 SET statement sets values corresponding to the occurrence numbers indicated by operands after the TO phrase to each operand before the TO phrase. The operand after the TO phrase is called the "sending operand" and the operand before the TO phrase is called the "receiving operand."
2. Combinations of sending operands and receiving operands in the SET statement are listed in the table below.

Sending Operand		Receiving Operand		
		index-name-1	identifier-1	
			Index Data Item	Integer Item
index-name-2		o (5)(a)	o (5)(b)	o (5)(c)
integer-2	index data item	o (5)(a)	o (5)(b)	-
	integer item	o (5)(a)	-	-
integer-1		o (5)(a)	-	-

o: Combined

-: Not combined

(n): General rule number

3. When index-name-1 is specified, the value of the operand after the TO phrase must correspond to a valid occurrence number of the table to which index-name is related.
4. When index-name-2 is specified, the value of index-name-2 must correspond to a valid occurrence number of the table to which index-name is related.
5. Index values are set according to the following rules:
    a. When index-name-1 is specified, the value of the occurrence number corresponding to index-name-2, identifier-2 or integer-1 is moved to index-name-1. When an index data item is specified in identifier-2, or when the index-name related to the same table as index-name-1 is specified, the value of the occurrence number is not converted.
    b. When an index data item is specified in identifier-1, it is set to the value of the contents of index-name-2 or identifier-2 without being converted.
    c. When an integer item is specified in identifier-1, the value of the occurrence number corresponding to the value of index-name-2 is set in identifier-1.

6. When two or more receiving operands exist, each is set to the initial value of index-name-2 or identifier-2. When any identifier-1 contains subscripts, the subscripts are evaluated in sequence for each identifier-1, after any effect the SET may have had on the value of the subscript.

Rules for Format 2

1. A format 2 SET statement does either of the following:
   a. Increases the value of each operand to the left of the UP BY phrase by the value of the operand to the right of the UP BY phrase.
   b. Decreases the value of each operand to the left of the DOWN BY phrase by the value of the operand to the right of the DOWN BY phrase.
2. The value of index-name-3 before and after execution of the SET statement must correspond to a valid occurrence number in the table to which index-name-3 is related.
3. When more than one index-name-3 has been written, the values of each index-name is increased or decreased by the initial the value for identifier-3.

Rules for Format 3

1. A format 3 SET statement sets the external switch related to each mnemonic-name-1 to ON or OFF.
2. The status of the external switch related to mnemonic-name-1 is set as follows:
   a. When the ON phrase is specified, the value of the external switch is set so that the condition name related the ON condition of the external switch is evaluated as true.
   b. When the OFF phrase is specified, the value of the external switch is set so that the condition name related the OFF condition of the external switch is evaluated as true.

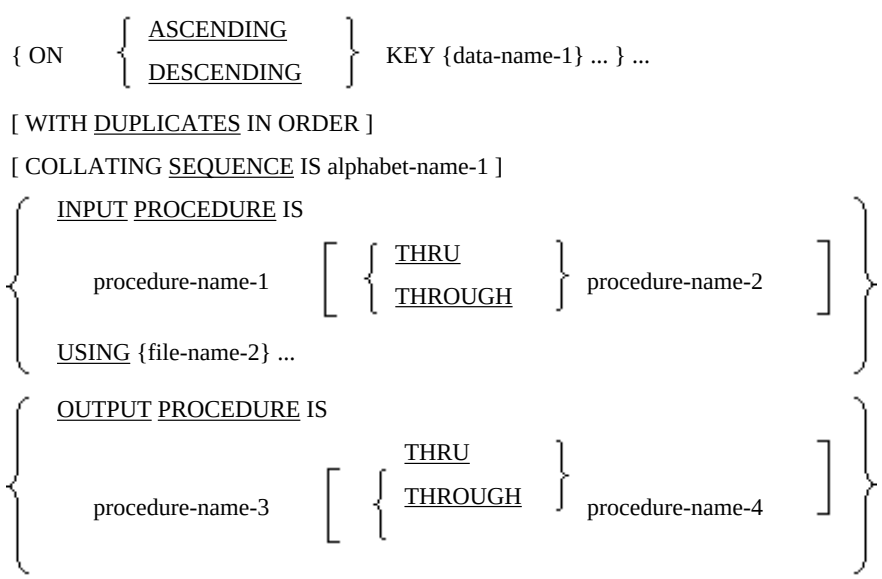
Rules for Format 4

1. A format 4 SET statement sets the value of the condition variable associated with condition-name-1 to the value specified in the VALUE clause of condition-name-1.
2. When two or more literals are specified in the VALUE clause of condition-name-1, the first value in the clause is used to set the condition variable.

6.4.41 SORT Statement (Sort-merge)

The SORT statement sorts records in the order determined by the key item values.

Format



GIVING {file-name-3}

Syntax Rules

1. The SORT statement can be written in any part of the procedure division except declaratives portion.
2. The file-name-1 must be defined in a Sort-Merge file description entry (SD) in the DATA DIVISION.
3. The following rules apply to file-name-2:
   a. If the record format of file-name-1 is variable, the record length of file-name-2 must be greater than or equal to the minimum record length and less than or equal to the maximum record length of file-name-1.
   b. If the record format of file-name-1 is fixed, the record length of file-name-2 must be less than or equal to the length of file-name-1.
4. The following rules apply to data-name-1:
   a. Data-name-1 must be a data item defined in a record description entry of file-name-1.
   b. Data-name-1 can be qualified.
   c. Data-name-1 cannot be a group item followed by a variable occurrence data item.
   d. When two or more record description entries have been defined for file-name-1, a data item in one record description entry must be specified in data-name-1. The character position that is the same as the data item of data-name-1 in the record description entry is regarded as the key to all records in the file.
   e. Data-name-1 cannot be a data item specifying an OCCURS clause or a data item following the data item which specifies an OCCURS clause.
   f. If the record format of file-name-1 is variable, data-name-1 must be included in the range of the minimum record size of file-name-1.
   F g. The data-name-1 cannot be a Boolean data item.
   h. See Appendix B, "System Quantitative Restrictions" for the maximum number of data-name-1's can be specified.
5. File-name-2 and file-name-3 must be defined in a file description entry, not in a sort-merge file description entry, in the DATA DIVISION.
6. File-name-2 and file-name-3 can be files on the same multiple file reels.
7. No pairs of file-name-1, file-name-2, and file-name-3 in the same SORT statement can be specified in one SAME AREA or one SAME SORT/SORT-MERGE AREA clause. File-names associated with the GIVING clause cannot be specified in the same SAME clause.
8. THROUGH is synonymous with THRU.
9. The following rules apply when file-name-3 is an indexed file:
   a. The first KEY phrase must be the ASCENDING KEY phrase.
   b. The character position in the record of the data item specified as the first data-name-1 must be the same as the position of the prime record key in the indexed file. Both data items must have the same length.
10. The following rules apply to file-name-3:
    a. When the record format of file-name-3 is variable, the record length of file-name-1 must be greater than or equal to the minimum record length of file-name-3 and less than or equal to the maximum record length.
    b. When the record format of file-name-3 is fixed, the record length of file-name-1 must be less than or equal to the length of file-name-3.
11. F Alphabet-name-1 must not be an alphabet-name associated to any of the following names in the ALPHABET clause in the SPECIAL-NAMES paragraph.
    - function-name

- SJIS
- UTF8
- UTF16
- UTF16LE
- UTF16BE
- UTF32
- UTF32LE
- UTF32BE

Specifying SJIS, UTF8, UTF16, UTF16LE, UTF16BE, UTF32, UTF32LE or UTF32BE in the ALPHABET clause is specific to [Win32], [Winx64] and [Linux64].

12. The CHARACTER TYPE or PRINTING POSITION clause must not be specified in the record description entry of file-name-1, file-name-2 or file-name-3.
13. File-name-2 and file-name-3 must be files as follows:
    a. Sequential files on a mass storage unit or floppy disk drive
    b. Relative files in sequential access mode
    c. Indexed files in sequential access mode
14. See Appendix B, "System Quantitative Restrictions" for the maximum number of file-name-2's that can be specified.

General Rules

Rules for the SORT Statement

1. When the SORT statement is executed, the following processing is performed:
    a. Records are delivered to the sorting operation. This processing is called "the first step of the sorting operation".

       The input procedure is executed when an INPUT PROCEDURE phrase is specified. The INPUT PROCEDURE clause indicates a procedure defined by procedure-name-1 or a procedure defined by the procedures procedure-name-1 and procedure-name-2. When the RELEASE statement is executed the contents of the file-name-1 record are delivered to the sorting operation.

       When the USING phrase is specified, a READ statement is executed for file-name-2 and the input record is delivered to the sorting operation. Before beginning this step, the file assigned to file-name-2 must not be open. During the execution of this step, the file assigned to file-name-2 is automatically opened and closed.

    b. Records in file-name-1 are sorted in the order specified in the KEY phrase. This processing is called "the sorting operation". File-name-2 and file-name-3 are not processed in this step.

    c. Records sorted become usable as records of file-name-1. This operation is called the "last step of the sorting operation". They are processed as follows:

       - The output procedure is executed when an OUTPUT PROCEDURE clause is specified. The OUTPUT PROCEDURE clause indicates the procedure defined by procedure-name-3 or the procedures defined in the procedures of procedure-name-3 to procedure-name-4. When the RETURN statement has been executed a record of file-name-1 becomes available for the output procedure.

       - When the GIVING phrase is specified, sorted records are written to the file assigned to file-name-3. Before starting this step, the file assigned to file-name-3 must not be open. During operation of this step, the file assigned to file-name-3 is automatically opened and closed.

2. Data-name-1 is called a "key item". To specify two or more key items, specify key items from left to right in the order determined by the desired key hierarchy.

    a. When the ASCENDING phrase is specified, key fields of the records delivered to the sorting operation are compared according to the collating sequence rules and returned in order from the smallest key value to the largest

    b. When the DESCENDING phrase is specified, key fields of the records delivered to the sorting operation are compared according to the collating sequence rules and returned in order from the largest key value to the smallest.

3. When comparison described in (2) finds two or more records containing the same contents in all key items, sorted records are returned as follows:

    a. If the DUPLICATES phrase is specified:

    If the USING phrase is specified, duplicate records in each file are returned in the order in which they were read. If the INPUT PROCEDURE phrase is specified, records are returned in the order that they were released.

    b. When the DUPLICATES phrase has been omitted:

    The order of returning records within the duplicate keys is undefined.

4. The sorting order of the key item values is determined according to the following rules:

    a. When the PROGRAM COLLATING SEQUENCE phrase is specified, records are sorted by the collating sequence corresponding to alphabetic-name-1.

    b. When the PROGRAM COLLATING SEQUENCE clause is omitted, records are sorted by the operating system's native collating sequence.

Rules for the INPUT PROCEDURE Phrase

1. When the INPUT PROCEDURE phrase is specified, control is transferred to the input procedure before the sorting operation starts. After the last statement in the input procedure has been executed, the sorting operation starts.
2. A RELEASE statement for a record of file-name-1 must be included in the input procedure. Each execution of the RELEASE statement delivers one record to the first step of the sorting operation.
3. The statements executed in the input procedure are called the "range of the input procedure". The range of the input procedure is as follows:

    a. Statements written in the input procedure

    b. All statements that are executed as the result of a transfer of control by executing statements in the range of input procedure (e.g. CALL or PERFORM statement)

    c. All statements in declarative procedures executed by statements in the input procedure

4. The MERGE, RETURN, and SORT statements must not be executed by the input procedure.
5. Any EXIT PROGRAM statement specified in the procedure division containing the SORT statement must not be executed by the input procedure.

Rules for the USING Phrase

1. When the USING phrase is specified, all records in the file assigned to file-name-2 are transferred to the file assigned to file-name-1.
2. For each file file-name-2, the following processing is performed:

    a. The initial processing is performed. This processing is the same as the execution of an OPEN statement with the INPUT phrase for the file assigned to file-name-2.

    b. Records in file-name-2 are delivered to the sorting operation. This processing is equivalent to repeatedly executing a READ statement with the NEXT and AT END phrases specified. When file-name-2 is a relative file, the contents of relative key items after the SORT statement execution are unusable as keys.

c. Termination processing is performed. This processing is equivalent to the execution of a CLOSE statement for file-name-2. This processing is performed before the records are actually sorted.

During processing from (a) to (c), a USE AFTER STANDARD EXCEPTION procedure may be executed. If the USE procedure is executed, the USE procedure must not execute any I-O statements to the file assigned to file-name-2 or execute any statement that references any records of file-name-2.

3. If the record format of file-name-1 is fixed and the record length of file-name-2 is less than the record length of file-name-1, the record is padded with blanks on the right when it is delivered to the file assigned to file-name-1.

Rules for the OUTPUT PROCEDURE Phrase

1. When the OUTPUT PROCEDURE phrase is specified, control transfers to the output procedure after the sorting operation terminates. After the last statement in the output procedure has been executed, control transfers to the next executable statement after the SORT statement.
2. When control transfers to the output procedure, records in file-name-1 are sorted and ready to be returned by the RETURN statement. A RETURN statement for the file assigned to file-name-1 must be specified in the output procedure. Each execution of the RETURN statement receives one record from the last step of the sorting operation.
3. The statements executed in the output procedure are called the "range of the output procedure." The range of the output procedure is as follows:
    a. Statements written in the output procedure
    b. All statements that are executed as the result of a transfer of control by executing statements in the range of output procedure (e.g. CALL or PERFORM statement)
    c. All statements in declarative procedures executed by statements in the output procedure
4. The MERGE, RETURN, and SORT statements must not be executed in the output procedure.
5. Any EXIT PROGRAM statement specified in the procedure division containing the SORT statement must not be executed by the output procedure.

Rules for the GIVING Phrase

1. When the GIVING phrase is specified, all records in the file assigned to file-name-1 are transferred to the file assigned to file-name-3 after termination of the sorting operation.
2. For each file-name-3, the following processing is performed:
    a. The initial processing is performed. This processing is equivalent to the execution of an OPEN statement with the OUTPUT phrase for the file assigned to file-name-3.
    b. Records in file-name-1 are returned from the sorting operation. This processing is equivalent to repeatedly executing a READ statement from file-name-1 and a WRITE statement to file-name-3 until there are no more sorted records available.

        If file-name-3 is a relative file, the relative key item is incremented from 1 by 1 each time a record is returned. The contents of the relative key item after execution of the SORT statement is the number of the last record returned to the file assigned to file-name-3.
    c. The termination processing is performed. This processing is the equivalent to the execution of a CLOSE statement that specifies file-name-3.

    During processing from (a) to (c), the USE AFTER STANDARD EXCEPTION procedure may be executed. When executing the USE AFTER STANDARD PROCEDURE, the procedure must not execute any I-O statements to the file assigned to file-name-3 or execute any statements that refer to records of file-name-3.
3. When an attempt to write the first record to the file assigned to file-name-3 is made, the processing is as follows:
    a. If a USE AFTER STANDARD EXCEPTION procedure related to file-name-3 is defined, the USE procedure is performed. After the control returns from the USE procedure, termination processing is performed as if a CLOSE statement specifying file-name-3 had been executed.

b. When a USE AFTER STANDARD EXCEPTION procedure related to file-name-3 is not defined, termination processing is performed as if a CLOSE statement specifying file-name-3 had been executed.

4. If the record format of file-name-3 is fixed and the record length of file-name-1 is less than the record length of file-name-3, the right side of the record is space filled when it is output to the file assigned to file-name-3.

6.4.42 START Statement (Relative File)

The START statement logically positions a file for accessing records sequentially.

Format

START file-name-1

[KEY { IS EQUAL TO / IS = / IS GREATER THAN / IS > / IS NOT LESS THAN / IS NOT < / IS GREATER THAN OR EQUAL TO / IS >= } data-name-1]

[INVALID KEY imperative-statement-1]

[NOT INVALID KEY imperative-statement-2]

[END-START]

Note

Symbols =, >, <, and >= are required, but not underlined to avoid confusion with other symbols.

Syntax Rules

1. File-name-1 must be a file opened in sequential or dynamic access mode.
2. Data-name-1 must be the data item (a relative key item) specified in the RELATIVE KEY phrase of the ACCESS MODE clause in the file control entry related to file-name-1. Data-name-1 can be qualified.
3. When a USE AFTER STANDARD EXCEPTION procedure related to file-name-1 is not defined, the INVALID KEY phrase must be specified.

F

In this implementation of COBOL, however, both the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.

General Rules

1. The file assigned to file-name-1 must be opened in input mode or I-O mode before executing the START statement.
2. When the KEY phrase has been omitted, the relation IS EQUAL TO is assumed as the default.
3. When the START statement is executed, the contents of the record area of file-name-1 are not changed.

4. The key related to the record in the file assigned to file-name-1 and contents of the relative key item in file-name-1 are compared in accordance with the condition specified in the KEY phrase. The rules for numeric comparison are applied. As a result of the comparison, the following processing will be performed:
    a. If a record that satisfies the comparison condition exists in the file assigned to file-name-1, the file position indicator is set in the first record satisfying the comparison condition.
    b. If no record satisfying the comparison condition exists in the file assigned to file-name-1, an invalid key condition occurs and execution of the START statement fails.
5. Execution of the START statement updates the value of the I-O status of the file assigned to file-name-1.
6. If file-name-1 is not an optional file, and if the file does not exist, execution of the START statement causes an invalid key condition and execution of the START statement fails.
7. If execution of the START statement fails, a value indicating that no valid record remains is moved to the file position indicator.
8. If an invalid key condition occurs during execution of the START statement, a value indicating an invalid key condition is moved to the I-O status of file-name-1, and control transfers according to the rule described in the topic titled "INVALID KEY Phrase" in the "Common Statement Rules" section earlier in this chapter.
9. If no other exceptional condition occurs during execution of the START statement, a value indicating an invalid key condition is moved to the I-O status of file-name-1 and control transfers according to the rule described in the topic titled "INVALID KEY Phrase" of the "Common Statement Rules" section earlier in this chapter.
10. The END-START phrase terminated the range of a START statement.
11. When AUTOMATIC is specified in the LOCK MODE phrase of the file related to file-name-1, the existing record is released from the lock when the START statement is executed.

6.4.43 START Statement (Indexed File)

The START statement logically positions a file for accessing records sequentially.

Format 1

Files are logically positioned to be accessed in the normal ascending sequence.

```
START file-name-1
                    IS EQUAL TO
[                   IS =                                  ]
[                   IS GREATER THAN                       ]
[                   IS >                                  ]
[   KEY          {  IS NOT LESS THAN                   }  {data-name-1}...  ]
[                   IS NOT <                              ]
[                   IS GREATER THAN                       ]
[                              OR EQUAL TO                ]
                    IS >=
[INVALID KEY imperative-statement-1]
[NOT INVALID KEY imperative-statement-2]
[END-START]
```

F

Format 2

Files are logically positioned to be accessed in descending sequence.

START file-name-1

```
                        IS EQUAL TO
                        IS =
                        IS LESS THAN
                        IS <
[ KEY                   IS NOT GREATER THAN          {data-name-1}... ]
                        IS NOT <
                        IS LESS THAN
                                OR EQUAL TO
                        IS <=
WITH REVERSED ORDER
[INVALID KEY imperative-statement-1]
[NOT INVALID KEY imperative-statement-2]
[END-START]
```

F

Format 3

Files are logically positioned to a record which is at the beginning of a specific key of reference.

```
START file-name-1
    FIRST
    [KEY IS {data-name-1} ...]
    [WITH REVERSED ORDER]
    [INVALID KEY imperative-statement-1]
    [NOT INVALID KEY imperative-statement-2]
    [END-START]
```

Note

The symbols =, >, <, and >= are required, but not underlined to avoid confusion with other symbols.

Syntax Rules

1. File-name-1 must be a file opened in sequential or dynamic access mode.
2. Data-name-1 can be qualified.
3. If the USE AFTER STANDARD EXCEPTION procedure related to file-name-1 is not defined, the INVALID KEY phrase must be specified.

   F

   In this implementation of COBOL, however, both the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.
4. If the KEY phrase is specified, data-name-1 must be one of the following.

   F

   However, if a format 3 START statement is written specifying the KEY phrase, data-name-1 must adhere to rule (a) below.

   a. Data-name-1 must be a prime record key item that was specified in the RECORD KEY clause of the file control entry related to file-name-1 or alternate record key item specified in the ALTERNATE RECORD KEY clause.

      F

      When two or more data-names are specified in the RECORD KEY clause of the file control entry related to file-name-1, some, or all, of the same data-names, in the same order, must be specified in the KEY clause of

the START statement. In the same way, when two or more data-names have been specified in the ALTERNATE RECORD KEY clause of the file control entry related to file-name-1 and the alternate key is used to START the file, some, or all, of the same data-names, in the same order, must be specified in the KEY clause of the START statement.

F

b. When only one data-name is specified in the RECORD KEY clause related to file-name-1, data-name must satisfy all of the following conditions:

- Data-name must exist in the record related to file-name-1 and the left most character position must equal the left most character position of the prime record key item.
- The size of data-name must be less than or equal to the size of the prime record key item.
- Data-name must be an alphanumeric data item, national data item, or group item.
- Only one data-name can be specified in the KEY clause.

c. When only one data-name is specified in the ALTERNATE RECORD KEY clause related to file-name-1, data-name must satisfy all of the following conditions:

d. Data-name must exist in the record related to file-name-1 and the left most character position must equal that of the left most character position of the alternate record key item.

- The size of data-name must be less than or equal to the size of the alternate record key item.
- Data-name must be an alphanumeric data item, national data item, or group item.
- Only one data-name can be specified in the KEY clause.

General Rules

Rules Common to All Formats

1. The file assigned to file-name-1 must be opened in input or I-O mode before executing the START statement.
2. Execution of the START statement does not update the contents of the record area of file-name-1, and the contents of the data item specified in the DEPENDING ON phrase of RECORD clause related to file-name-1 is not updated.
3. Execution of the START statement updates the value of the I-O status of the file of file-name-1.
4. If file-name-1 is not an optional file, and does not exist, execution of the START statement causes an invalid key condition and execution of the START statement fails.
5. If execution of the START statement fails, a value indicating that no more valid records exist is moved to the file position indicator. The key of reference is undefined.
6. The key of reference is determined according to the following rules which are applied in the order described:

   a. When the KEY phrase is omitted, the prime record key of file-name-1 is the key of reference

F

   b. When all names specified by data-name-1 of the KEY clause are the same as, and specified in the same order as the names specified in the RECORD KEY clause of the file control entry of file-name-1, the prime record key of file-name-1 is the key of reference.

   c. When all names specified by data-name-1 of the KEY clause are the same as, and specified in the same order as the names specified in the ALTERNATE RECORD KEY clause of the file control entry of file-name-1, the alternate record key of file-name-1 is the key of reference.

   d. When some of the names specified by data-name-1 of the KEY clause are the same as, and specified in the same order as the names specified in the RECORD KEY clause of the file control entry of file-name-1, the list of data-name-1's becomes the key of reference.

   e. When some of the names specified by data-name-1 of the KEY clause are the same as, and specified in the same order as the names specified in the ALTERNATE RECORD KEY clause(s) of the file control entry of file-name-1, the key of reference is determined subject to the following conditions.

   - When data-name-1 matches part of the alternate record key specification in one ALTERNATE RECORD KEY clause, it becomes the key of reference.

- When data-name-1 matches part of the alternate record key specification in two or more ALTERNATE RECORD KEY clauses, the alternate record key consisting of the minimum number of data-names is the key of reference. If the minimum number of data-names exist among two or more alternate record keys, the first alternate record key in which the ALTERNATE RECORD KEY clause is specified in the file control entry of file-name-1 is the key of reference.

f. If the data-name specified by data-name-1 is not a data-name that was specified in the RECORD KEY or ALTERNATE RECORD KEY of the file control entry associated with file-name-1, the key of reference is the record key that satisfies all the following conditions.

- The left end character position equals that of the data item in data-name-1.

- The size of data-name is larger than or equals that of the data item in data-name-1.

F

- The record key consists of only one data item.

- Data-name is an alphanumeric data item, national data item, or group item.

g. When two or more record keys that satisfy all the conditions in (f) exist, the following record key will be the key of reference.

- If the prime record key satisfies condition (f), the prime record key is the key of reference.

- If the prime record key does not satisfy condition (f), the smallest alternate record key among alternate record keys to satisfy the condition will be the key of reference. If two or more alternate record keys are equally smallest, the first alternate record key defined by the ALTERNATE RECORD KEY clause in the file control entry of file-name-1 is the key of reference.

7. The key of reference determined in accordance with rule (6) is used to determine the order of record keys in the START statement. When the START statement has executed successfully, the key of reference specified in the START statement will be used for subsequent READ statements in the sequential access mode.

F

8. When a START statement is executed successfully, the searching method for subsequent READ statements in sequential access mode for the file assigned to file-name-1 is determined as follows:

   - For a START statement without the REVERSED ORDER phrase, the searching operation will be performed in the normal ascending key order.
   - For a START statement with the REVERSED ORDER phrase, the searching operation will be performed in the descending key order.

9. If an invalid key condition occurs during execution of a START statement, a value indicating an invalid key condition is moved to the I-O status of file-name-1 and control transfers according to the rules described in the topic titled "INVALID KEY Phrase" of the "Common Statement Rules" section earlier in this chapter.

10. If execution of the START statement is successful, the I-O status of file-name-1 is updated and control transfers according to the rules described in the topic titled "NOT INVALID KEY Phrase" in the "Common Statement Rules" section earlier in this chapter.

11. The END-START phrase terminates the range of the START statement.

12. If the AUTOMATIC clause is specified in the LOCK MODE phrase of the file related to file-name-1, the existing record is released from the lock when a START statement is executed.

Rules Common to Format 1 and Format 2

1. According to the description of the KEY phrase, the record keys existing in the file assigned to file-name-1 are compared with data items as follows:

   a. When the KEY phrase is specified, the data item of data-name-1 is compared with the record key.

   b. When the KEY phrase is omitted, the data item specified in the RECORD KEY clause of file-name-1 is compared with the record key of the records in the file. The relation "IS EQUAL TO" is assumed in the KEY phrase.

2. If the sizes of the keys in the file and the keys described in the KEY phrase differ, the rightmost end of the longer operand is truncated to be the same length as the shorter one, and the characters are compared. As a result of comparison:

   a. In format 1, the file position indicator is set to the value indicated by the first record satisfying the comparison condition.

F b. In format 2, the file position indicator is set to the value indicated by the last record satisfying the comparison condition.

   c. If no record in the file satisfies the condition above, an invalid key condition occurs and execution of the START statement fails.

F Rules for Format 3

1. When a format 3 START statement is executed, the file position indicator of file-name-1 is updated as follows:

   a. If the REVERSED ORDER phrase is omitted, the file assigned to file-name-1 is positioned at the first record of the key of reference. The file position indicator is set to the value of the key of reference.

   b. If the REVERSED ORDER phrase is specified, the file assigned to file-name-1 is positioned at the last record of the key of reference. The file position indicator is set to the value of the key of reference.

   c. If no valid record exists in the file, an invalid key condition occurs and the START statement execution fails.

2. Data-name-1 is specified only to determine the key of reference.

Example

Example for selection of the key of reference

```
ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
    FILE-CONTROL.
      SELECT IX-FS1 ASSIGN TO SYS006
        ORGANIZATION IS INDEXED
        RECORD KEY IS MKEY1 MKEY2                  *> [A]
        ALTERNATE RECORD KEY IS IKEY1 IKEY2 IKEY3  *> [B]
        ALTERNATE RECORD KEY IS IKEY2 IKEY3 IKEY4  *> [C]
        ALTERNATE RECORD KEY IS IKEY1 IKEY3        *> [D]
        ALTERNATE RECORD KEY IS IKEY3 IKEY4 IKEY5  *> [E]
        ALTERNATE RECORD KEY IS IKEY3 IKEY4        *> [F]
        ALTERNATE RECORD KEY IS IKEY3 IKEY5.       *> [G]
DATA DIVISION.
 FILE SECTION.
  FD  IX-FS1.
  01  IX-FS1R1-F-G-240.
    02 IX-FS1-GRP-120.
      03 MKEY1    PIC X(5).
      03 MKEY2    PIC X(5).
      03 IKEY1    PIC X(29).
      03 IKEY2    PIC S9(9).
      03 IKEY3    PIC X(10).
      03 IKEY4    PIC X(100).
      03 IKEY5    PIC 9(7).
PROCEDURE DIVISION.
    START IX-FS1.                          *> Rule 6a : the key of reference is [A]
    START IX-FS1 KEY IS = MKEY1 MKEY2.     *> Rule 6b : the key of reference is [A]
    START IX-FS1 KEY IS = IKEY1 IKEY2 IKEY3.
                                           *> Rule 6c : the key of reference is [B]
    START IX-FS1 KEY IS = MKEY1.           *> Rule 6d : the key of reference is [A]
    START IX-FS1 KEY IS = IKEY1 IKEY2.     *> Rule 6e : the key of reference is [B]
    START IX-FS1 KEY IS = IKEY3.           *> Rule 6e : the key of reference is [F] (*1)
```

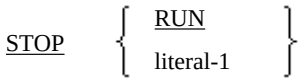
*1:

- Because IKEY3 is not written the first, B, C, and D are excluded from the selection.
- Because the number of data that composes E is more than that of F and G, E is excluded from the selection.
- In F and G, F previously written is selected.

6.4.44 STOP Statement (Nucleus)

The STOP statement terminates execution of a run unit. The STOP statement specifying literal-1 is an obsolete element.

Format



Syntax Rules

1. Literal-1 must not be
   - a figurative constant starting with the keyword ALL
   - a national nonnumeric literal.

F

2. If literal-1 is a numeric literal, it must be an unsigned integer.

F

   In this implementation of COBOL, however, literal-1 or a literal with a symbol or a decimal point can be specified.

3. If the STOP RUN statement is written in a series of imperative statements of one sentence, the STOP RUN statement must be the last statement in the series of imperative statements.

General Rules

1. When a STOP RUN statement is executed, the executing run unit terminates and the control is transferred to the operating system.
2. For those files that are open at the execution of a STOP RUN statement, an implicit CLOSE statement without any optional phrases is executed. USE procedures associated with these files are not executed.
3. If a STOP RUN statement that specifies literal-1 is executed, execution of the run unit is suspended and the value of literal-1 is reported to the operator. If the operator instructs the program to continue execution, execution restarts with the next executable statement following the STOP statement.

6.4.45 STRING Statement (Nucleus)

The STRING statement links character-strings.

For details about how to use the examples, refer to "STRING Statement" in the "Syntax Samples".

Format 1

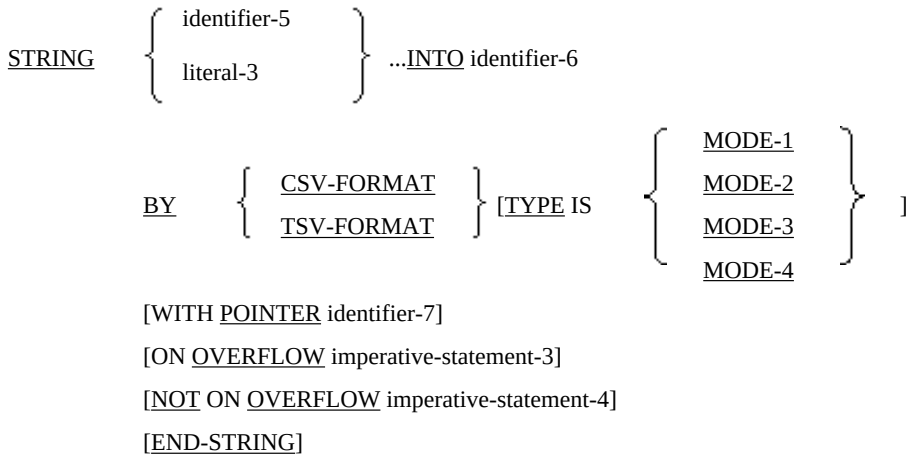
INTO identifier-3 [WITH POINTER identifier-4]

[ON OVERFLOW imperative-statement-1]

[NOT ON OVERFLOW imperative-statement-2]

[END-STRING]

Format 2



Note

Format2 is available for [Win32], [Winx64], [Solaris], [Linux64] and [.NET].

Syntax Rules

Rules for format 1

1. Literal-1 and literal-2 must be
   - nonnumeric literals,
   - national nonnumeric literals,
   - or figurative constants not starting with the keyword ALL.
2. Identifier-1, identifier-2, and identifier-3 must be used for presentation items.
3. If identifier-1 and identifier-2 are numeric data items, they must be integer items that do not contain PICTURE symbol P.
4. Identifier-3 must not be
   - an alphanumeric edited data item,
   - national edited data item,
   - or numeric edited data item.

   Identifier-3 must not be a data item in which the JUSTIFIED clause is specified.
5. Identifier-3 cannot be reference modified.
6. Identifier-4 must be an integer item not containing PICTURE character P. Identifier-4 must be large enough to hold the value of identifier-3 plus one character.
7. If the data category of identifier-1, identifier-2, identifier-3, literal-1, or literal-2 is a National or national edited data item, all categories must be National or national edited data items.
8. The encoding form of identifier-1 to identifier-3 must be the same. This rule is specific to [Win32], [Winx64] and [Linux64].

Rules for format 2

1. The DECIMAL POINT IS COMMA clause cannot be used.
2. Literal-3 must be a nonnumeric literal, a national nonnumeric literal, or a figurative constant SPACE.

3. Identifier-5 must be a data item with USAGE DISPLAY or a group item containing only data items. Identifier-5 must not specify the following:
   - ANY LENGTH clause
   - REDEFINES clause
   - BLANK WHEN ZERO clause
   - CHARACTER TYPE clause
   - JUSTIFIED clause
   - OCCURS clause with the DEPENDING ON phrase
   - PRINTING POSITION clause
   - SYNCHRONIZED clause
   - TYPE clause
4. Identifier-6 must be an alphanumeric character, not a data edited data item. The following cannot be used:
   - JUSTIFIED clause
   - ANY LENGTH clause
5. Identifier-7 must be large enough to hold the value of identifier-6 plus one character.
6. When identifier-5 or the elementary item subordinate to identifier-5 is a numeric data item, it must be an integer item not containing the PICTURE character P.

General Rules

Rules for format 1

1. The STRING statement combines the character-strings of identifier-1 or literal-1 and transfers the result to identifier-3. Identifier-1 and literal-1 are called "sending items". Identifier-3 is called a "receiving item". When a STRING statement is executed, all or part of the character-string of each sending item is transferred to the high order end character position of the receiving item or to the character position following the position specified in identifier-4.

   To transfer all the character-strings of the sending item, specify the DELIMITED BY phrase. To transfer a set of characters from the high order end character position of the sending item up to a certain character, specify identifier-2 or literal-2 in the DELIMITED BY phrase. Identifier-2 and literal2 are called "delimiters".

   To transfer sending items to a position following a specific character position specified in the receiving item, move the character position of the target character position of the receiving item to identifier-4, and specify identifer-4 in the POINTER phrase. The value of identifier-4 is incremented by one character position each time a sending item character is transferred to the receiving item.

   If the POINTER phrase is omitted, the sending item is transferred to the first character position of the receiving item.

2. In accordance with the description in the DELIMITED BY phrase, the following character-strings in sending items are transferred:
   a. When identifier-2 or literal-2 is specified in the DELIMITED BY phrase, the character-string from the high order end of the sending item to the character immediately preceding the delimiter is transferred. The delimiter itself is not transferred.
   b. When the SIZE is specified in the DELIMITED BY phrase, the entire character-string of the sending items are transferred.
3. The initial value of identifier-4 must be set before executing the STRING statement. The initial value must be greater than 1. The initial value of identifier-4 is used as the initial value of the "current transfer position". The "current transfer position" is incremented by 1 each time a transfer is performed and it is moved to identifier-4 when the STRING statement execution terminates.

4. If the POINTER phrase is omitted, the initial value of the "current transfer position" is 1.

5. One character each is transferred from each sending item to the receiving item according to the transfer rules. A character-string is transferred from each sending item to the receiving item by repeating the following processing:

   a. When the "current transfer position" does not exceed the number of characters of the receiving item, one character in the sending item is transferred to the current transfer position.

   b. When the "current transfer position" exceeds the number of characters of the receiving item, an overflow condition occurs and the transfer operation terminates. Control then transfers in accordance with the description of the ON OVERFLOW phrase.

   c. The processing of (a) or (b) repeats until the delimiter specified in the DELIMITED BY phrase is detected (if identifier-2 or literal-2 is specified in the DELIMITED BY phrase), until all data in the sending item has been transferred, or until data is transferred up to the end of the receiving items.

6. Only the transferred part of the receiving item is updated by execution of a STRING statement. Other parts of the receiving item remain unchanged by execution of a STRING statement.

7. When a figurative constant is specified as a delimiter, the figurative constant is regarded as

   - one-nonnumeric literal or

F - national nonnumeric literal.

8. After the transfer operation of the STRING statement is executed, transfer of control is controlled by the presence or absence of the ON OVERFLOW or NOT ON OVERFLOW phrases. Transfer of control after the transfer operation is illustrated in the table below.

ON OVERFLOW phrase	NOT ON OVERFLOW phrase	Control transfer	
		Overflow condition occurred in the transfer operation. (*3)	No overflow condition occurred in the transfer operation. (*2)
Yes	Yes	The transfer operation terminates and control transfers to imperative-statement-1. After imperative-statement-1 has been executed, control transfers to the end of the STRING statement. (*1)	After all the character-string has been transferred, control transfers to imperative-statement-2. After imperative-statement-2 has been executed, control transfers to the end of the STRING statement.
Yes	No	The transfer operation terminates and control transfers to imperative-statement-1. After imperative-statement-1 has been executed, control transfers to the end of the STRING statement. (*1)	After all character-strings have been transferred, control transfers to the end of the STRING statement.
No	Yes	The transfer operation terminates and control transfers to the end of the STRING statement.	After all the character-string has been transferred, control transfers to imperative-statement-2. After imperative-statement-2 has been executed, control transfers to the end of the STRING statement.
No	No	The transfer operation terminates and the control transfers to the end of the STRING statement.	After all character-strings have been transferred, control transfers to the end of the STRING statement.

*1 If a procedure branching or conditional statement causes an explicit transfer of control in imperative-statement-1, control transfers according to the statement rules.

*2 If a procedure branching or conditional statement causes an explicit transfer of control in imperative-statement-2, control transfers according to the statement rules.

*3 The overflow condition occurs if:

- The initial value of identifier-4 is less than 1 or more than the number of characters in the character-string of identifier-3.
- The "current transfer position" is larger than the number of characters in the character-string of identifier-3. That is, some characters have been left in the sending item without being transferred even though character-strings have been transferred up to the low order end of identifier-3.

9. The END-STRING phrase terminates the range of the STRING statement.
10. The result of executing a STRING statement is undefined in the following cases:
    a. Identifier-1 or identifier-2 has the same storage area as identifier-3 or identifier-4.
    b. Identifier-3 has the same storage area as identifier-4.
    c. Identifier-1, identifier-2 or identifier-3 is a group item, and encoding forms of subordinate data items are different.

Rules for format 2

1. Identifier-5 and literal-3 are called "sending items". Identifier-6 is called a "receiving item".
2. When literal-3 uses a figurative constant, it is regarded as a single digit data item.
3. The BY specification is applied after the character string stored in identifier-5 or literal-3 is processed by the STRING statement. The result is stored in identifier-6. When identifier-5 is a group item, it sequentially stores the character string starting with the first subordinate data item.
    - Trailing blanks are deleted on alphanumeric or national characters.
    - For numerics, all leading zeros and any trailing zeros in the decimal part are removed.
4. When the classes and the encoding forms of the sending and receiving sides are different, data is converted according to the class and the encoding forms of the receiving side. When an error is detected, an overflow condition occurs.
5. The data format is specified using the BY specification.
    - When CSV-FORMAT is specified, identifier-6 is converted to CSV as specified by TYPE. CSV is character data that is delimited by commas.
    - When TSV-FORMAT is specified, identifier-6 is converted to TSV format as specified by TYPE. TSV is character data that is delimited by tabs.
6. The data format is specified using the TYPE phrase. Refer to "Variation of CSV" in Chapter 26 of the NetCOBOL User's Guide for details of MODE-n.
7. If the TYPE phrase is omitted, MODE-1 is assumed as the default.
8. If the POINTER phrase is specified, the initial value of identifier-7 must be set before executing the STRING statement. The initial value must be greater than 1.
9. If the POINTER phrase is omitted, the initial value of identifier-7 is 1.
10. One character each is transferred from each sending item to the receiving item according to the transfer rules. A character-string is transferred from each sending item to the receiving item by repeating the following processing:
    - When the value of the POINTER (identifier-7) does not exceed the number of characters of the receiving item (identifier-6), one character in the sending item is transferred to the receiving item (identifier-6).
    - When the POINTER (identifier-7) exceeds the number of characters of the receiving item (identifier-6), an overflow condition occurs and the transfer operation terminates. Control then transfers in accordance with the description of the ON OVERFLOW phrase.
11. The processing above repeats until the delimiter specified in the BY phrase is detected, until all data in the sending item has been transferred, or until data is transferred up to the end of the receiving items.

12. Only the transferred part of identifier-6 is updated by the execution of a STRING statement. Other parts of identifier-6 remain unchanged by the execution of a STRING statement.

13. After the transfer operation of the STRING statement is executed, transfer of control is controlled by the settings of ON OVERFLOW and NOT ON OVERFLOW as illustrated in the table below.

ON OVERFLOW phase	NOT ON OVERFLOW phase	Control transfer	
		Overflow condition occurred in the transfer operation. (*3)	No overflow condition occurred in the transfer operation. (*2)
Yes	Yes	The transfer operation terminates and control transfers to imperative-statement-3. After imperative-statement-3 has been executed, control transfers to the end of the STRING statement. (*1)	After all character-strings have been transferred, control transfers to imperative-statement-4. After imperative-statement-4 has been executed, control transfers to the end of the STRING statement.
Yes	No	The transfer operation terminates and control transfers to imperative-statement-3. After imperative-statement-3 has been executed, control transfers to the end of the STRING statement. (*1)	After all character-strings have been transferred, control transfers to the end of the STRING statement.
No	Yes	The transfer operation terminates and control transfers to the end of the STRING statement.	After all character-strings have been transferred, control transfers to imperative-statement-4. After imperative-statement-4 has been executed, control transfers to the end of the STRING statement.
No	No	The transfer operation terminates and the control transfers to the end of the STRING statement.	After all character-strings have been transferred, control transfers to the end of the STRING statement.

*1 If a procedure branching or conditional statement causes an explicit transfer of control in imperative-statement-3, control transfers according to the statement rules.

*2 If a procedure branching or conditional statement causes an explicit transfer of control in imperative-statement-4, control transfers according to the statement rules.

*3 The overflow condition occurs if:

- The initial value of identifier-7 is less than 1, or more than the number of characters in the character-string of identifier-6.
- An error was detected when moving to the receiving item.
- Identifier-7 is larger than identifier-6.

14. The END-STRING phrase terminates the STRING statement.

15. When executing a STRING statement, the result is undefined in the following cases:

- Identifier-5 has the same storage area as identifier-6 or identifier-7.
- Identifier-6 has the same storage area as identifier-7.

Processing Flow of a STRING Statement

An example of the processing flow of a STRING statement is demonstrated below.

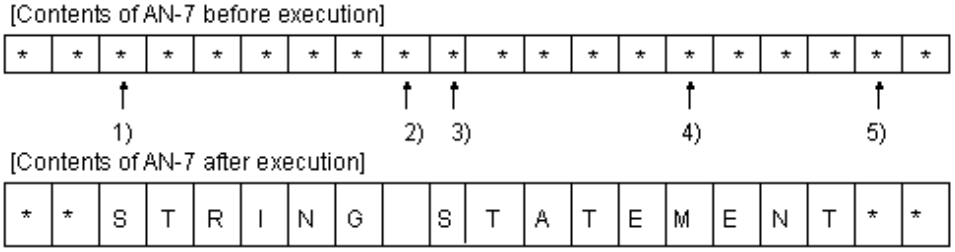
[DATA DIVISION]

```
 77  AN-1  PIC X(6)   VALUE "STRING".
 77  AN-2  PIC X(5)   VALUE "STATE".
 77  AN-4  PIC X(6)   VALUE "MENT#0".
 77  AN-5  PIC X(4)   VALUE "#123".
 77  AN-7  PIC X(20)  VALUE "********************".
 77  ED-8  PIC 99     VALUE 3.
```

[PROCEDURE DIVISION]

```
    STRING AN-1 SPACE AN-2 DELIMITED BY SIZE
           AN-4 AN-5 DELIMITED BY "#"
                INTO AN-7 WITH POINTER ED-8.
```

When this STRING statement is executed, sending items (AN-1, SPACE, AN-2, AN-4 and AN-5) are linked sequentially and transferred to the receiving item (AN-7).



- The initial value of ED-8 is 3, therefore position 3 and all subsequent characters of AN-7 will be the target for transferring.
- The entire character-string of AN-1 is transferred first. As a result of transfer, the value of ED-8 is updated to 9.
- Then, the figurative constant SPACE is transferred as one nonnumeric literal. As a result of transfer, the value of ED-8 is updated to 10.
- The entire character-string of AN-2 is transferred next. As a result of transfer, the value of ED-8 is updated to 15.
- The character-string up to # in AN-4 is transferred. As a result of transfer, the value of ED-8 is updated to 19.
- Since there is no character strings preceding the # in AN-5, no character-string is transferred. The value of ED-8 is not updated.
- The value of ED-8 is 19 when the STRING statement execution terminates.

6.4.46 SUBTRACT Statement (Nucleus)

The SUBTRACT statement obtains the result of subtraction.

Format 1

The target of the subtraction is replaced with the minuend.

SUBTRACT { identifier-1 / literal-1 } ...

FROM { identifier-2 [ROUNDED] } ...

[ON SIZE ERROR imperative-statement-1]

[NOT ON SIZE ERROR imperative-statement-2]

[END-SUBTRACT]

Format 2

The result of the subtraction is stored in a specified data item.

```
SUBTRACT { identifier-1 } ...
         { literal-1    }
    FROM { identifier-2 }
         { literal-2    }  GIVING { identifier-3 [ ROUNDED ] } ...
  [ ON SIZE ERROR imperative-statement-1 ]
  [ NOT ON SIZE ERROR imperative-statement-2 ]
  [ END-SUBTRACT ]
```

Format 3

Subtraction is performed by corresponding the data items following two group items.

```
SUBTRACT { CORRESPONDING }
         { CORR          }
  identifier-1 FROM identifier-2 [ ROUNDED ]
  [ ON SIZE ERROR imperative-statement-1 ]
  [ NOT ON SIZE ERROR imperative-statement-2 ]
  [ END-SUBTRACT ]
```

Syntax Rules

1. In formats 1 and 2, identifier-1 and identifier-2 must be numeric data items. In format 3, identifier-1 and identifier-2 must be group items.
2. Identifier-3 must be a numeric data item or numeric edited data item.
3. Literal-1 and literal-2 must be numeric literals.
4. ORRESPONDING is synonymous with CORR.

General Rules

Rules for Format 1

A format 1 SUBTRACT statement subtracts the operands preceding the FROM phrase from the identifiers following the FROM phrase and stores the result in identifier-2. The subtraction is performed in the order in which identifier-2 is written.

Rules for Format 2

A format 2 SUBTRACT statement in subtracts the operands preceding the FROM phrase from the operands following the FROM phrase and stores the result in identifier-3. The subtraction is performed in the order in which identifier-3 is written.

Rules for Format 3

A format 3 SUBTRACT statement subtracts subordinate data items of the group items preceding the FROM phrase from subordinate data items that have the same names and qualifiers of the group items following the FROM phrase. The subordinate data items of identifier-1 are treated as the subtrahend and the subordinate data items of identifier-2 that have the same names and qualifiers as those in identifier-1 are treated as the minuend in order to obtain the differences, which are stored in the corresponding subordinate the data item of identifier-2. The result is as though one or more individual SUBTRACT statements have been written for each subordinate data item in identifier-1 and identifier-2 that have the same names and qualifiers.

Rules Common to All Formats

1. The END-SUBTRACT phrase terminates the range of the SUBTRACT statement.

2. See the section titled "Common Statement Rules" earlier in this chapter for the rules of the ROUNDED phrase, ON SIZE ERROR phrase, CORRESPONDING phrase, the operation, and the transfer.

6.4.47 SUPPRESS Statement (Report Writer)

The SUPPRESS statement suppresses presentation of a report group.

The report writer module cannot be used in [Winx64], [LinuxIPF] and [.NET].

Format

SUPPRESS PRINTING

Syntax Rule

The SUPPRESS statement can be written only in a USE BEFORE REPORTING procedure.

General Rules

1. The only report groups suppressed by the SUPPRESS statement are the report groups specified in the USE procedure containing the SUPPRESS statement.
2. To suppress all report groups, the SUPPRESS statement must be executed for each group.
3. When the SUPPRESS statement is executed, the report writer control system suppresses the following:
    a. Presentation of print lines in the report group
    b. Processing for all LINE NUMBER phrases in the report group
    c. Processing for the NEXT GROUP phrase in the report group
    d. Line-counter updating
    e. Any page advance associated with the report group

6.4.48 TERMINATE Statement (Report Writer)

The TERMINATE statement terminates report processing.

The report writer module cannot be used in [Winx64], [LinuxIPF] and [.NET].

Format

TERMINATE {report-name-1} ...

Syntax Rules

Report-name-1 must be specified in a report description entry of report section in the DATA DIVISION.

General Rules

1. When the TERMINATE statement is executed, the report writer control system writes and displays all control footing groups in the ascending order beginning from the lowest level. Then, the report writer control system writes and displays the report footing report groups. The report writer control system enables the old values of control data items

which have been stored previously to be used in the SOURCE phrase or a USE procedure of the control footing group and the report footing report group, just as though the system detected the control break of the control data-name in the highest level.

2. If a TERMINATE statement is executed after execution of an INITIATE statement, and no GENERATE statement has been executed, the report writer control system does not write and display any report group or perform any processing related to the operation.
3. When a page feed is required for displaying the report body group during the presentation of the report, the report writer control system automatically performs page feed processing if a page heading report group and a page footing group have been defined.
4. A TERMINATE statement for report-name-1 must not be executed when an INITIATE statement has not been executed for report-name-1 or when report-name-1 has already been terminated by execution of a previous TERMINATE statement of report-name-1.
5. The result of executing a TERMINATE statement that specifies two or more report-name-1s is equivalent to sequentially executing multiple TERMINATE statements that specify each report-name-1.
6. The TERMINATE statement does not close the files related to the report. A CLOSE statement must be executed separately to close a file after execution of the TERMINATE statement. If report-name-1 has been started by execution of an INITIATE statement, a TERMINATE statement must be executed for report-name-1.

F

6.4.49 UNLOCK Statement (Sequential File, Relative File, Indexed File)

The UNLOCK statement cancels all locks obtained for file connectors.

Format

```
UNLOCK file-name-1 { RECORD  }
                   { RECORDS }
```

General Rules

1. File-name-1 must be a file opened by execution of a successful OPEN statement, regardless of whether it is locked or not.
2. When the UNLOCK statement is executed, all locks obtained by the file connector for file-name-1 are canceled.
3. If no lock exists in the file assigned to file-name-1, the UNLOCK statement is executed successfully.

6.4.50 UNSTRING Statement (Nucleus)

The UNSTRING statement separates character-strings into sub-strings.

For details about how to use the examples, refer to "UNSTRING Statement" in the "Syntax Samples".

Format 1

```
UNSTRING identifier-1
[ DELIMITED BY [ALL] { identifier-2 } [ OR [ALL] { identifier-3 } ] ]
                     { literal-1   }             { literal-2   }
    INTO {identifier-4 [DELIMITER IN identifier-5]
                       [COUNT IN identifier-6]}...
    [WITH POINTER identifier-7]
    [TALLYING IN identifier-8]
    [ON OVERFLOW imperative-statement-1]
```

[NOT ON OVERFLOW imperative-statement-2]

[END-UNSTRING]

Format 2

UNSTRING identifier-9 INTO identifier-10 ...

BY { CSV-FORMAT / TSV-FORMAT }

[WITH POINTER identifier-11]

[TALLYING IN identifier-12]

[ON OVERFLOW imperative-statement-3]

[NOT ON OVERFLOW imperative-statement-4]

[END-UNSTRING]

Note

Format2 is available for [Win32], [Winx64], [Solaris], [Linux64] and [.NET].

Syntax Rules

Rules for Format 1

1. Literal-1 and the literal-2 must be nonnumeric literals, national nonnumeric literals(*), or figurative constants that do not start with the keyword ALL.
2. Identifier-1, identifier-2, identifier-3, and identifier-5 must be alphanumeric data items, national data items(*), or national edited data items(*).
3. Identifier-4 must be alphabetic data item, alphanumeric data item, national data item(*) or numeric data item.

   The numeric data item is used for presentation and must be a character-string not containing PICTURE character "P".
4. Identifier-6 and identifier-8 must be integers that do not contain the PICTURE character "P".
5. Identifier-7 must be an integer that does not contain the PICTURE character "P". Identifier-7 must be large enough to hold the data item identifier-1 plus one character.
6. Identifier-1 cannot be reference modified.
7. When the category of data of identifier-1 to identifier-5, literal-1, or literal-2 is a national or national edited data item, all categories must be national or national edited data items.
8. The encoding form of identifier-1 to identifier-5 must be the same. This rule is specific to [Win32], [Winx64] and [Linux64].

Rules for Format 2

1. The DECIMAL POINT IS COMMA clause cannot be specified.
2. Identifier-9 must be an alphanumeric character, not a data edited data item. The following cannot be used:
   - JUSTIFIED clause
   - ANY LENGTH clause
3. Identifier-10 must be a data item with USAGE DISPLAY or group items containing only data items, and must not contain the PICTURE character P. Identifier-10 must not specify the following:
   - ANY LENGTH clause

- REDEFINES clause
- BLANK WHEN ZERO clause
- CHARACTER TYPE clause
- JUSTIFIED clause
- OCCURS clause with the DEPENDING ON phrase
- PRINTING POSITION clause
- SYNCHRONIZED clause
- TYPE clause

4. Identifier-11 must be an integer item not containing the PICTURE character P, and it must be large enough to hold the value of identifier-9 plus one character.
5. Identifier-12 must be an integer item not containing the PICTURE character P.

General Rules

Rules for Format 1

1. The UNSTRING statement selects character strings from the contents of identifier-1 according to the conditions specified in the DELIMITED BY phrase and transfers the strings to identifier-4. Identifier-1 is called a "sending item". Identifier-4 is called a "receiving item". Identifier-2, identifier-3, literal-1, and literal-2 are called "delimiters".

   An UNSTRING statement with the DELIMITED BY phrase controls the number of character positions of the data to transfer. Data is transferred, one character at a time, from the sending item to the receiving item beginning with the first sending character position (See POINTER, below.). The sending item is examined, beginning with the first sending character position, character by character, until the character specified in the DELIMITED BY phrase is encountered, or until the maximum length of the sending item is reached. If the character specified by the DELIMITED BY phrase is found in the sending item, the character number of the character preceding the DELIMITED BY character becomes the ending position for the transfer. If the character specified in the DELIMITED BY phrase is not encountered in the sending item, the ending position is the length of the sending item.

   An UNSTRING statement without the DELIMITED BY phrase transfers data, in the order written, from the sending item to the receiving item beginning with the first sending character position (See POINTER, below.) for the number of character positions to transfer. The number of character positions to transfer of the sending item becomes the length the receiving item(s).

   When data is selected from the beginning of the sending item, the POINTER phrase is omitted. When is selected from a specific character position in the sending item, the POINTER phrase is specified and the starting character position is moved to identifier-7 in the POINTER phrase. The value of identifier-7 is incremented each time it is compared with the delimiter.

   The number of characters transferred to each receiving item, up to the delimiter, specify the COUNT IN phrase. To obtain the position of the delimiter, specify the DELIMITER IN phrase.

   The number of receiving items of which character-strings have been actually transferred can be counted. To count the number of receiving items into which character-strings have been actually transferred, specify the TALLYING IN phrase.

2. The initial value of identifier-7 must be initialized before executing the UNSTRING statement. The initial value must be greater than 1. The initial value of identifier-7 is used as an initial value of the "current test position". The value of the "current test position" increments by 1 each time a character in the sending item is tested. The current position when the UNSTRING statement execution terminates becomes the value of identifier-7.
3. When the POINTER phrase is omitted, the initial value of the "current test position" is 1.

4. When the DELIMITED BY phrase is specified, the sending items are selected as follows and transferred to each receiving item. The first "current receiving item" is the first identifier-4 specified in the INTO phrase in the following description.

    a. When the "current test position" does not exceed the number of characters of the sending item, one character in the sending item following the "current test position" is compared with the delimiter. When two or more delimiters have been specified, the character-strings starting from the same position are compared repeatedly in the order that delimiters have been specified until they match the delimiters. When the delimiter contains two or more characters, the character-string containing the same number of characters as that of the delimiter is compared with the delimiter. When the "current test position" exceeds the number of characters of the sending item, the delimiter test terminates.

    b. If the character matching the delimiter is found at the position after the "current test position" in the processing (a), the character-string from the "current test position" to the character preceding the character matching the delimiter is transferred to the "current receiving item." The delimiter is not transferred. When the DELIMITER IN phrase is specified, the delimiter is transferred to identifier-5.

    c. If no character matching the delimiter is found in processing (a), the character-string in the sending item is transferred to the "current receiving item". A space is transferred to identifier-5 specified in the DELIMITER IN phrase.

    d. If the character at the "current test position" matches the delimiter in processing (a), the following values are transferred to the "current receiving items".

        - When the "current receiving item" is an alphabetic data item, or alphanumeric data item, a space is transferred.

        - When the "current receiving item" is a national data item(*), a space is transferred.

        - When the "current receiving item" is a numeric data item, zero is transferred.

    e. When the COUNT IN phrase is specified, the number of characters transferred to the "current receiving item" is moved to identifier-6. In the case of (d), identifier-6 is set to zero.

    f. The "current test position" and the "current receiving item" are changed as follows and the processing (a) to (f) repeats until all characters in the sending items are transferred or all receiving items are used.

        - The character position to the right of the character matching the delimiter becomes the "current test position."

        - The next receiving item is the "current receiving item."

5. When the DELIMITED BY phrase is omitted, the sending items are divided as in the following procedure and transferred to the receiving items. The initial value of the "current receiving item" in the following processing is the first identifier-4 in the INTO phrase.

    a. When the "current test position" does not exceed the number of characters in the sending items, the number of characters in the sending items after the "current test position" is compared with the number of characters in the "current receiving item." When the "current receiving item" is a numeric data item, the following value is used as the number of characters.

        - When the SEPARATE is specified in the SIGN clause, the number of characters excluding the number of operational signs is used.

        - When the decimal point is specified in the PICTURE clause, the number of characters excluding the number of decimal points is used.

    b. When the "current test position" exceeds the number of characters in the sending items, the transfer operation terminates.

    c. A character-string containing only the number of characters of the "current receiving item" starting from the "current test position" is transferred to the "current receiving item". Then, the "current test position" and the "current receiving item" are changed as follows and the processing (a) to (f) repeats until all characters in the sending items are transferred or all receiving items are used.

        - The character position to the right of the last character transferred becomes the "current test position."

- The next receiving item is the "current receiving item."

6. When the TALLYING phrase is specified, the number of character-strings that have actually been transferred to receiving items is added to identifier-8. Identifier-8 must be initialized before execution of the UNSTRING. The value of identifier-8 is incremented by one in each identifier-4.

7. When a figurative constant is specified as the delimiter, the figurative constant is regarded as a one-character nonnumeric literal.

8. When the ALL phrase is specified before the delimiter, multiple, sequential delimiters are skipped. When the DELIMITER IN phrase is specified, one copy of the delimiter is placed in identifier-5 regardless of the number of skipped characters.

9. Any character in the computer character set can be used as a delimiter.

10. After the UNSTRING statement transfer operation is executed, transfer of control depends on whether the ON OVERFLOW or NOT ON OVERFLOW phrases are specified. The transfer of control after the transfer operation is shown in the table below.

ON OVERFLOW phrase	NOT ON OVERFLOW phrase	Control transfer	
		Overflow condition occurred in the transfer operation. (*3)	No overflow condition occurred in the transfer operation. (*2)
Yes	Yes	1. The transfer operation terminates and control transfers to imperative-statement-1. 2. After imperative-statement-1 has been executed, control transfers to the end of the UNSTRING statement. (*1)	1. After all of the character-string has been transferred, control transfers to imperative-statement-2. 2. After imperative-statement-2 has been executed, control transfers to the end of the UNSTRING statement.
Yes	No	1. The transfer operation terminates and control transfers to imperative-statement-1. 2. After imperative-statement-1 has been executed, control transfers to the end of the UNSTRING statement. (*1)	After all character-strings have been transferred, control transfers to the end of the UNSTRING statement.
No	Yes	The transfer operation terminates and control transfers to the end of the UNSTRING statement.	1. After all the character-string has been transferred, control transfers to imperative-statement-2. 2. After imperative-statement-2 has been executed, control transfers to the end of the UNSTRING statement.
No	No	The transfer operation terminates and control transfers to the end of the UNSTRING statement.	After all character-strings have been transferred, control transfers to the end of the UNSTRING statement.

*1 If a procedure branching or conditional statement causes an explicit transfer of control in imperative-statement-1, control transfers according to the statement rules.

*2 If a procedure branching or conditional statement causes an explicit transfer of control in imperative-statement-2, control transfers according to the statement rules.

*3 An overflow condition occurs in the following cases:

- The initial value of identifier-4 is less than 1 or greater than the number of characters in the character-string of identifier-3.
- During execution of an UNSTRING statement, untested characters are left in data item identifier-3 even though all receiving items have been used.

11. The END-UNSTRING phrase terminates the range of the UNSTRING statement.
12. The result of the UNSTRING statement execution is undefined in the following cases:
    a. Identifier-1, identifier-2, and identifier-3 have the same storage area as identifier-4 to identifier-8.
    b. Identifier-4, identifier-5, and identifier-6 have the same storage area as identifier-7 or identifier-8.
    c. Identifier-7 has the same storage area as identifier-8.

Rules for Format 2

1. Identifier-9 is called a "sending item". Identifier-10 is called a "receiving item".
2. Identifier-11 shows the character position in the area of identifier-9.
3. Identifier-12 is a counter for data items received into identifier-10.
4. Data is transferred, in the order written, from the sending item to the receiving item beginning with the first sending character position (see POINTER below). The sending item is examined per the format described by the BY specification. When identifier-10 is a group item, delimited items are transcribed one at a time, starting with the first subordinate data item.
5. Data is transcribed according to the class and the encoding form of the receiving side item when the class and the encoding form of the sending item and the receiving item are different. When errors are detected, an overflow condition occurs.
6. The BY specification specifies the delimited form. When CSV-FORMAT is specified, identifier-9 is divided by Comma Separated Value. Comma Separated Value is data delimited by commas. When TSV-FORMAT is specified, identifier -9 is divided by TSV format. TSV data delimited by tabs. In the compiler, characters are treated as follows:
    - Quotation marks are removed from the sending item data character string, and it is transferred to the receiving item. Each double quotation mark encountered in the data is replaced by a single quotation mark.
    - When two consecutive commas are detected in the sending item, it is transferred to the receiving as zeros if it is a figure, and blanks if it is an alphanumeric character or Japanese.
7. When the POINTER phrase is specified, the character string of identifier-9 is inspected starting from the position indicated by identifier-11. If the POINTER phrase is not specified, the character string is inspected starting from the first character position.
8. The user should set the initial value of POINTER (identifier-11) and TALLYING (identifier-12).
9. The value of the POINTER (identifier-11) is incremented each time a character is transferred to the receiving item (identifier-9).
10. The TALLYING phrase tallies how many items are found between delimiters.
11. After the UNSTRING statement transfer operation is executed, transfer of control depends on whether the ON OVERFLOW or NOT ON OVERFLOW phrases are specified. The transfer of control after the transfer operation is shown in the table below.

ON OVERFLOW phrase	NOT ON OVERFLOW phrase	Control transfer	
		Overflow condition occurred in the transfer operation. (*3)	No overflow condition occurred in the transfer operation. (*2)
Yes	Yes	1. The transfer operation terminates and control transfers to imperative-statement-1.	1. After all of the character-string has been transferred, control transfers to imperative-statement-2.

ON OVERFLOW phrase	NOT ON OVERFLOW phrase	Control transfer	
		Overflow condition occurred in the transfer operation. (*3)	No overflow condition occurred in the transfer operation. (*2)
		2. After imperative-statement-1 has been executed, control transfers to the end of the UNSTRING statement. (*1)	2. After imperative-statement-2 has been executed, control transfers to the end of the UNSTRING statement.
Yes	No	1. The transfer operation terminates and control transfers to imperative-statement-1. 2. After imperative-statement-1 has been executed, control transfers to the end of the UNSTRING statement. (*1)	After all character-strings have been transferred, control transfers to the end of the UNSTRING statement.
No	Yes	The transfer operation terminates and control transfers to the end of the UNSTRING statement. However, the control moves at the end of UNSTRING statement after the post is continued when it is (d) of overflow condition (*3), and all the posts end.	1. After all the character-string has been transferred, control transfers to imperative-statement-2. 2. After imperative-statement-2 has been executed, control transfers to the end of the UNSTRING statement.
No	No	The transfer operation terminates and control transfers to the end of the UNSTRING statement. However, the control moves at the end of UNSTRING statement after the post is continued when it is (d) of overflow condition (*3), and all the posts end.	After all character-strings have been transferred, control transfers to the end of the UNSTRING statement.

*1 If a procedure branching or conditional statement causes an explicit transfer of control in imperative-statement-3, control transfers according to the statement rules.

*2 If a procedure branching or conditional statement causes an explicit transfer of control in imperative-statement-4, control transfers according to the statement rules.

*3 An overflow condition occurs in the following cases:

- The initial value of identifier-11 is less than 1 or greater than the number of characters in the character-string of identifier-9.
- During execution of unstring statement, there are untested characters left in the data item identifier-9 even though all receiving items have been used.
- An error occurred when data was transferred to the receiving item.
- Data was omitted in the transfer to the receiving item.

12. The END-UNSTRING phrase terminates the range of the UNSTRING statement.
13. The result of the UNSTRING statement execution is undefined in the following cases:
    - Identifier-9 has the same storage area as identifier-10, Identifier-11, or identifier-12.
    - Identifier-11 has the same storage area as identifier-12.

F

* : Extended functions or functions specific to NetCOBOL.

- national nonnumeric literal
- national edited data item
- national data item

Processing Flow of the UNSTRING Statement

An example of the processing flow of the UNSTRING statement is given below.

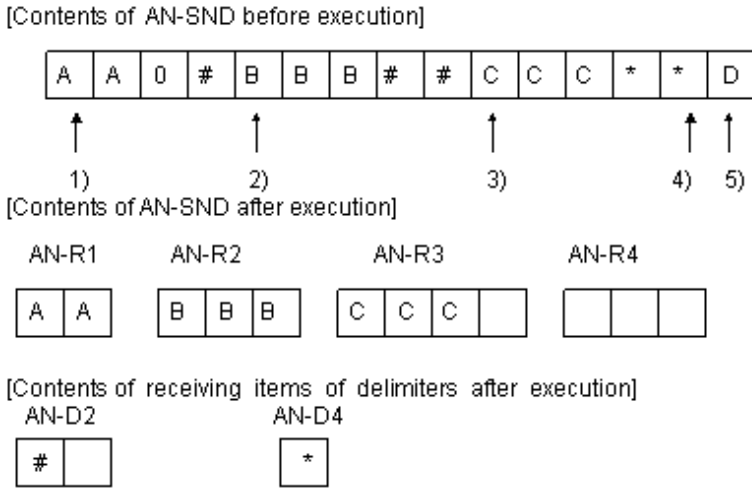
[DATA DIVISION]

```
 77  AN-SND PIC X(17) VALUE "AA0#BBB##CCC**DDD".
 77  AN-DEL PIC X     VALUE "*".
 77  AN-R1  PIC X(2)  VALUE "II".
 77  AN-R2  PIC X(3)  VALUE "JJJ".
 77  AN-R3  PIC X(4)  VALUE "KKKK".
 77  AN-R4  PIC X(3)  VALUE "LLL".
 77  AN-D2  PIC X(2)  VALUE "MM".
 77  AN-D4  PIC X     VALUE "N".
 77  ED-C3  PIC 99    VALUE 11.
 77  ED-C4  PIC 99    VALUE 22.
 77  ED-PTR PIC 99    VALUE  1.
 77  ED-TLY PIC 99    VALUE  0.
```

[PROCEDURE DIVISION]

```
    UNSTRING AN-SND
        DELIMITED BY ALL "#" OR AN-DEL
        INTO AN-R1
             AN-R2 DELIMITER IN AN-D2
             AN-R3 COUNT IN ED-C3
             AN-R4 DELIMITER IN AN-D4
                   COUNT IN ED-C4
        WITH POINTER ED-PTR
        TALLYING IN ED-TLY
        ON OVERFLOW GO TO OVERFLOW-PROC.
```

When this UNSTRING statement is executed, sending items (AN-SND) are delimited and transferred to the receiving item (AN-R1, AN-R2, AN-R3, and AN-R4).



- The initial value of ED-PTR is 1, so the first (position 1) and subsequent characters of AN-SND will be targets for transferring. The target character-strings are terminated by delimiters and transferred to the sending items. The delimiters are repetitive # and one-character *.
- The character-string starting from position 1 to the "#" is transferred to the AN-R1 first. As a result of transfer, the value of ED-PTR is changed to 5 (position 2).

- Then, the character-string starting from position 2 to the "#" is transferred AN-R2. As a result of transfer, the value of ED-PTR is changed to 10 (position 3).
- The character-string starting from position 3 to the "*" is transferred to AN-R3 next. As a result of transfer, the value of ED-PTR is updated to 14 (position 4). Since the number of characters transferred (up to the delimiter) is 3, the value of the ED-C3 is changed to 3.
- Since the character at position 4 is an "*", a blank is transferred to AN-R4. As a result of transfer, the value of ED-PTR is changed to 15 (position 5). Since the number of characters transferred (up to the delimiter) is 0, the value of the ED-C4 is changed to 0.
- The value of the ED-PTR is 15 when the UNSTRING statement execution terminates. Since there are characters that have not been not transferred left in AN-SND, control transfers to the OVERFLOW-PROC.
- Since character-strings have been transferred to four receiving items (AN-R1, AN-R2, AN-R3 and AN-R4), ED-TLY is incremented by 4 and the value of ED-TLY is changed to 4.

Example

Example of the UNSTRING Statement

Suppose that the data to be used in the UNSTRING statement is defined as follows:

```
    77 SEND-A PICTURE  X(30) VALUE "MUNICH/NEW!YORK//TOKYO********".
    77 RCV-1  PICTURE  X(10) VALUE SPACES.
    77 RCV-2  PICTURE  X(10) VALUE SPACES.
    77 RCV-3  PICTURE  X(10) VALUE SPACES.
    77 DEL-1  PICTURE  XX    VALUE SPACES.
    77 DEL-2  PICTURE  XX    VALUE SPACES.
    77 DEL-3  PICTURE  XX    VALUE SPACES.
    77 CTR-1  PICTURE  99    VALUE 0.
    77 CTR-2  PICTURE  99    VALUE 0.
    77 CTR-3  PICTURE  99    VALUE 0.
    77 PTR-A  PICTURE  99    VALUE 1.
    77 TALL-A PICTURE  99    VALUE 0.
```

Examples of UNSTRING statement description and the execution result are as follows:

Example 1

```
    UNSTRING SEND-A
        INTO RCV-1 RCV-2 RCV-3
        WITH POINTER PTR-A.
```

Identifier	Identifier value after execution
RCV-1	MUNICH/NEW
RCV-2	!YORK//TOK
RCV-3	YO********
PTR-A	31

Example 2

```
   UNSTRING SEND-A
       DELIMITED BY "/"
       INTO RCV-1 COUNT IN CTR-1
            RCV-2 COUNT IN CTR-2
            RCV-3 COUNT IN CTR-3
       WITH POINTER PTR-A TALLYING IN TALL-A.
```

Identifier	Identifier value after execution
RCV-1	MUNICH!!!!
RCV-2	NEW!YORK!!
RCV-3	!!!!!!!!!!
CTR-1	06
CTR-2	08
CTR-3	00
PTR-A	18
TALL-A	03

In this example, there are untested characters left in the SEND-A even though all receiving items have been used. If the ON OVERFLOW phrase is specified in the UNSTRING statement in this example, the imperative statement in the ON OVERFLOW phrase is executed.

Example 3

```
   UNSTRING SEND-A
       DELIMITED BY ALL "/" OR ALL "*"
       INTO RCV-1 DELIMITER IN DEL-1
                  COUNT IN CTR-1
            RCV-2 DELIMITER IN DEL-2
                  COUNT IN CTR-2
            RCV-3 DELIMITER IN DEL-3
                  COUNT IN CTR-3
       WITH POINTER PTR-A TALLYING IN TALL-A.
```

Identifier	Identifier value after execution
RCV-1	MUNICH!!!!
RCV-2	NEW!YORK!!
RCV-3	TOKYO!!!!!
DEL-1	/!
DEL-2	/!
DEL-3	*!
CTR-1	06
CTR-2	08
CTR-3	05
PTR-A	31
TALL-A	03

6.4.51 USE Statement (Sequential, Relative, Indexed, Presentation File, and Report Writer Module)

This statement specifies the start of an I-O error handling procedure.

For details about how to use the examples, refer to "USE Statement" in the "Syntax Samples".

Format 1

For sequential, relative, and indexed files

```
USE [GLOBAL] AFTER STANDARD
    { EXCEPTION }
    { ERROR     } PROCEDURE ON
    { {file-name-1} ... }
    { INPUT              }
    { OUTPUT             }
    { I-O                }
    { EXTEND             } .
```

F

Format 2

For presentation file

```
USE [GLOBAL] AFTER STANDARD
    { EXCEPTION }
    { ERROR     } PROCEDURE ON
    {  {file-name-1} ... }
    { INPUT              }
    { OUTPUT             }
    { I-O                } .
```

Format 3

For report file

```
USE [GLOBAL] AFTER STANDARD
    { EXCEPTION }
    { ERROR     } PROCEDURE ON
    { {file-name-1} ... }
    { OUTPUT             }
    { EXTEND             } .
```

Note

The report writer module cannot be used in [Winx64], [LinuxIPF] and [.NET].

Syntax Rules

The USE AFTER STANDARD EXCEPTION procedure is abbreviated to "USE procedure" in this explanation.

1. The USE statements must follow the DECLARATIVES section header of the PROCEDURE DIVISION and precede the END-DECLARATIVES statement. Only one declaratives section can be defined. It is followed by statements that define the USE procedures. Unless needed by the procedure logic, paragraph names are not required.
2. Each USE statement must have a unique file-name-1. Duplicate file-name-1 descriptions would cause multiple USE procedures to execute at the same time.
3. EXCEPTION and ERROR are synonymous.
4. Files having different ORGANIZATION or ACCESS MODE phrase values can be specified in a list of file-name-1 descriptions. Such files can also be implicitly referenced using the USE statement.

5. Only one USE statement can be specified in the declarative in one PROCEDURE DIVISION if it contains the INPUT, OUTPUT, I-O, or EXTEND phrase.

General Rules

1. The USE statement specifies the conditions that will cause execution of the USE procedure. The USE statement itself is not executed.
2. The USE procedure is executed if an input-output statement (including input-output statements executed implicitly) has failed, except when the AT END phase (sequential, relative, indexed, or presentation file) or INVALID KEY phase (relative or indexed file) is specified, and if so, these take precedence. Specify the conditions for executing the USE procedure as follows:
    a. The file-name-1 phrase executes the USE procedure if the input-output statement related to file-name-1 failed.
    b. The INPUT phrase specifies that the USE procedure is to be executed during execution of an input-output statement related to the file opened in input mode or if the file failed to be opened in input mode.
    c. The OUTPUT phrase specifies that the USE procedure is to be executed during execution of an input-output statement related to the file opened in output mode or if the file failed to be opened in output mode.
    d. The I-O phrase specifies that the USE procedure is to be executed during execution of an input-output statement related to the file opened in input-output mode or if the file failed to be opened in input-output mode.
    e. The EXTEND phrase specifies that the USE procedure is to be executed during execution of an input-output statement related to the file opened in extended mode or if the file failed to be opened in extended mode.
3. The declarative section can be written in any program from the innermost program to the outermost program.
4. A declarative section is selectively executed based on the priorities listed below if an input-output statement failed in a nested program. When a USE statement that satisfies specified conditions is found, only the USE procedure following the USE statement is executed.
    a. USE statement that is defined in a program containing the failed input-output statement and that satisfies the conditions.
    b. USE statement that has the GLOBAL phrase in a program directly or indirectly containing the failed input-output statement and that satisfies conditions. If more than one USE statement is found, the USE statement in the highest nested level program is selected.
5. Conditions specified in the USE statement are checked during execution of a separately compiled program containing the USE statement.
6. A USE procedure must not reference a procedure that is not a declaratives procedure.
7. A PERFORM statement must be used if a section-name in a declarative section or a paragraph name in a USE procedure are referenced in another declarative section or the procedure division.
8. When file-name-1 is defined in the USE statement, conditions specified in other USE statements do not apply to file-name-1.
9. If no serious error occurred in the I-O status, then, after execution of the USE procedure, control is transferred to the executable statement following the input-output statement that executed the USE procedure. For details on control transfer when a serious error occurs in the I-O status, refer to "NetCOBOL User's Guide"
10. Before the USE procedure is completed, no statement shall be executed which may cause the use procedure to be executed again.

6.4.52 USE BEFORE REPORTING Statement (Report Writer)

This statement specifies the start of the procedure that is executed immediately before a report group is created and displayed.

Format

USE BEFORE REPORTING identifier-1

Note

The report writer module cannot be used in [Winx64], [LinuxIPF] and [.NET].

Syntax Rules

1. The USE BEFORE REPORTING statement must follow the section header in the declarative portion of the PROCEDURE DIVISION. Only one USE statement can be defined for any one section header. It is followed by statements that define the USE BEFORE REPORTING procedure. Unless needed by the procedure logic, no paragraph names are required.
2. Identifier-1 must be a report group in the report section of the DATA DIVISION. The same identifier-1 must not be specified in more than one USE BEFORE REPORTING statement.
3. The GENERATE, INITIATE, or TERMINATE statement must not be written in a USE BEFORE REPORTING procedure. These statements must not be written in the range of execution of any PERFORM statement in the USE BEFORE REPORTING procedure.
4. The value of the control data items must not change in the USE BEFORE REPORTING procedure.

General Rules

1. The USE BEFORE REPORTING phrase specifies the conditions that will cause the execution of the USE BEFORE REPORTING procedure. The USE BEFORE REPORTING statement itself is not executed.
2. The declarative section can be written in any program from the innermost program to the outermost program.
3. The Report Writer Control System executes the USE BEFORE REPORTING procedure immediately before the report group specified in identifier-1 is created and displayed.
4. The USE BEFORE REPORTING procedure must not cause itself to be re-executed.
5. A PERFORM statement must be used when a section-name in the declarative section or a paragraph name in the USE BEFORE REPORTING procedure are referenced from another declarative section or the PROCEDURE DIVISION.

F

6.4.53 USE FOR DEAD-LOCK Statement

The USE FOR DEAD-LOCK statement specifies the procedure to be executed when a deadlock occurs.

Note

The USE FOR DEAD-LOCK statement can only be used in [DS], [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF] and [Linux64].

Format

USE FOR DEAD-LOCK

Syntax Rules

1. The USE FOR DEAD-LOCK statement must be written following the declaratives section header in the PROCEDURE DIVISION. The USE FOR DEAD-LOCK statement, as such, must be a sentence. After the USE FOR DEAD-LOCK statement, write statements that define the procedures to execute. Unless needed by the procedure logic, no paragraph name is required.
2. The USE FOR DEAD-LOCK statement itself is not executed, and only specifies the execution condition for the USE FOR DEAD-LOCK procedures.
3. The USE FOR DEAD-LOCK statement can only be written once in a program.

4. The USE FOR DEAD-LOCK statement cannot be written in a program that is contained in another program.

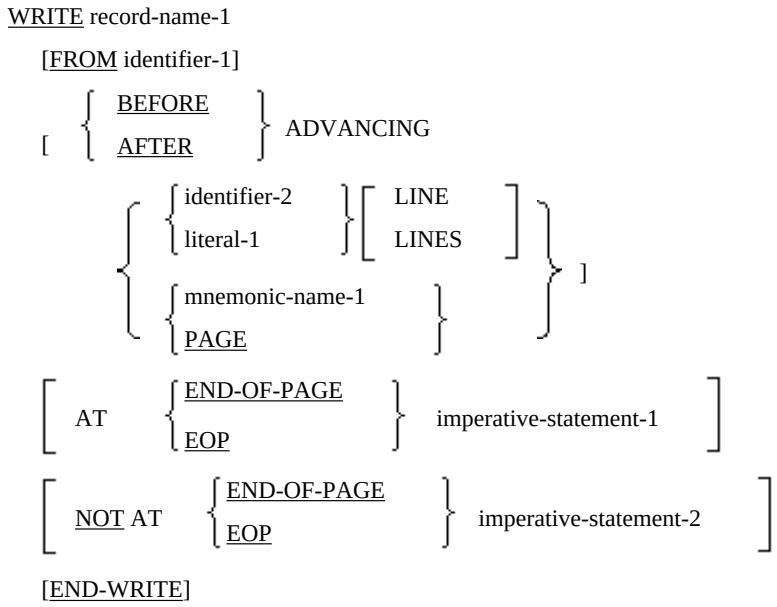
General Rules

1. The specified procedure is executed by the transaction control system.
2. A procedure-name associated with the USE FOR DEAD-LOCK statement can be referenced in another declarative section or in a procedure in the PROCEDURE DIVISION only by using the PERFORM statement.
3. After its execution, a USE FOR DEAD-LOCK procedure must reference a procedure that is not a declaratives procedure in a GO TO statement. When the control is transferred to the end of the USE FOR DEAD-LOCK procedure, the program is abnormally terminated.
4. At the termination of a USE FOR DEAD-LOCK procedure, the COBOL standard error processing procedure is executed before control returns to the PROCEDURE DIVISION.
5. During execution of a USE FOR DEAD-LOCK procedure, no statement may be executed that might cause the same USE FOR DEAD-LOCK statement to be executed again.
6. At termination of the DEAD-LOCK procedure, if a run unit contains two or more programs in that contain USE FOR DEAD-LOCK statements, control passes to the program that was executed most recently (of those programs having USE FOR DEAD-LOCK statements).
7. Refer to the "NetCOBOL User's Guide" for the action taken when a USE FOR DEAD-LOCK statement is written in a run unit.

6.4.54 WRITE Statement (Sequential File)

This statement writes records to a file or scrolls lines on a logical page.

Format



Syntax Rules

1. Record-name-1 must be the name of a record defined in the FILE SECTION of the DATA DIVISION. It can be qualified by the file name.
2. When a data item is specified in identifier-1, the same area must not be specified for record-name-1 and identifier-1. A function-identifier specified in identifier-1 must be an alphanumeric function.
3. Identifier-2 must be an integer item.

4. Integer-1 can be zero.
5. When the LINAGE clause is specified in a file description entry related to record-name-1, mnemonic-name-1 must not be specified.
6. Mnemonic-name-1 must be assigned to the function names CHANNEL-01 to CHANNEL-12, SLC, CTL, STACKER-01 or STACKER-2 in the SPECIAL-NAMES paragraph in the ENVIRONMENT DIVISION.
7. A WRITE statement must not contain both the ADVANCING PAGE phrase and END-OF-PAGE phrase.
8. When the END-OF-PAGE or NOT END-OF-PAGE phrase is specified, the LINAGE clause must be specified in the file description entry related to record-name-1.
9. END-OF-PAGE and EOP are synonymous.

F

10. The END-OF-PAGE phrase can be specified in the WRITE statement for a print file only. It cannot be specified in the WRITE statement for a print file having the FORMAT clause.
11. The ADVANCING phrase can only be specified in a WRITE statement for a print file.

    In [HP], [Win32], [Winx64], [Solaris], [LinuxIPF], [Linux64] and [.NET], the ADVANCING phrase can also be specified in a WRITE statement for a line sequential file.
12. A mnemonic name assigned to the function name CTL must not be specified in mnemonic-name-1 when the ADVANCING phrase is specified in the WRITE statement for the print file having the FORMAT clause.
13. The ADVANCING phrase must not be specified when the record description entry of record-name-1 specifies the CONTROL RECORDS clause.
14. Reference modification must not be made using identifier-1 when identifier-1 explicitly or implicitly contains the data item using the CHARACTER TYPE or PRINTING POSITION clauses.
15. In a WRITE statement for a printing file having the FROM phrase, the encoding form of identifier-1 must be same as the encoding form of record-name-1 specified explicitly or implicitly. This rule is specific to [Win32], [Winx64] and [Linux64].

General Rules

1. The file related to record-name-1 must have been opened in output or extended mode before execution of the WRITE statement.
2. After the WRITE statement has been executed successfully, the record specified in record-name-1 can no longer be used in the record area. However, execution of the WRITE statement does not make the record unusable if the file related to record-name-1 specified the SAME RECORD AREA clause in the I-O-CONTROL paragraph. If so, the program can reference record-name-1.
3. A WRITE statement having the FROM phrase produces the same result as the following statements executed in the provided order. The printing results comply with general rule 20.:
   a. Statements according to the MOVE statement rules

      MOVE identifier-1 TO record-name-1

   b. Same WRITE statement as above, but without FROM phrase
4. After the WRITE statement has been executed, the information of record-name-1 area cannot be used except in cases in which the file related to record-name-1 is specified in the SAME RECORD AREA clause. However, information in identifier-1 is usable.
5. The file position indicator remains unchanged after the WRITE statement is executed.
6. The value of the I-O status of the file related to record-name-1 is updated after execution of the WRITE statement.
7. The record is passed to the I-O control system during execution of the WRITE statement.
8. The size of the record in record-name-1 must not exceed the maximum record length of the file related to record-name-1. The size of the record in record-name-1 must not be less than the minimum record length of the file related to record-name-1. If these conditions are not satisfied, the WRITE statement will fail and no record is written. At this

time, the record area of the file related to record-name-1 remains unchanged. The I-O status of the file related to record-name-1 is updated with a value indicating the cause of the failure.

F Note that the compiler accepts the WRITE successfully even when a record smaller than the minimum record size is specified, since it cannot predict the runtime environment.

9. The END-WRITE phrase delimits the range of the WRITE statement.
10. The sequence of records in a sequential file is determined by the execution sequence of the WRITE statements that create the file. It is changed only when records are appended to the file.
11. F When records are added to a file opened in extended and shared mode with the WRITE statement for the file connector of two or more files, the order of record addition may be undefined.
12. When the file related to record-name-1 is opened in EXTENDED mode, a WRITE statement appends records as if the file were opened in output mode. If the file contains records, the WRITE statement adds records immediately after the last record that existed in the file when the OPEN statement was executed.
13. If the WRITE statement attempts to write records beyond the file size specified explicitly or implicitly from an external program, data in the record area remains unchanged, an exception condition occurs and processing is as follows.
    a. The I-O status of the file related to record-name-1 is updated.
    b. If a USE AFTER STANDARD EXCEPTION procedure corresponding to the file related to record-name-1 is defined, the procedure is executed. After execution of the procedure, control is passed according to the rules of the USE statement. Otherwise, control is passed to the end of the WRITE statement if the FILE STATUS clause is specified in the file related to record-name-1. If neither USE AFTER EXCEPTION procedure nor FILE STATUS phrase is specified, the results are undefined.
14. The program performs the following processing when the reel or unit file reaches its end and it is within an area defined externally:
    a. Executes the standard at-end reel or unit label procedure.
    b. Exchanges the reel or unit. That is, the volume indicator is updated to the next reel or unit in the file.
    c. Executes the standard start reel or unit label procedure.
15. The ADVANCING and END-OF-PAGE phrases for a print file perform vertical line control in the presentation of the print page. If the ADVANCING phrase is omitted, AFTER ADVANCING 1 LINE is assumed and the program automatically performs a line feed. When ADVANCING is specified, a line feed is performed as explained below.
    a. When a positive integer is specified in integer-1 or identifier-2, the specified number of lines is fed.
    b. When a negative value is specified in identifier-2, the result is undefined.
    c. When zero is specified in integer-1 or identifier-1, no line feed is made.
    d. When mnemonic-name-1 is specified, lines are skipped to channel 1 to 12 or the line feed is suppressed according to the function name assigned to mnemonic-name-1. Mnemonic-name-1 is assigned to the function name as shown below.
        - Function name SLC suppresses line feed.
        - Function names CHANNEL-01 to CHANNEL-12 skip lines to channel 1 to 12, respectively. Channel 1 skips to a new page.
        - Function name CTL writes a control record that defines the page format. For information on writing the control record, see general rule 19.
    e. The BEFORE phrase performs line feed according to rules (a) to (d), before outputting the lines.
    f. The AFTER phrase performs line feed according to rules (a) to (d), after outputting the lines.
    g. When the ADVANCING PAGE phrase is specified and the LINAGE clause is specified in the file description entry related to record-name-1, the statement writes a record to the logical page before (BEFORE) or after (AFTER) the column is set to the first line of the body in the next logical page specified in the LINAGE clause.

h. Otherwise, the statement writes a record to the physical page before (BEFORE) or after (AFTER) the column is set to the next physical page.

F i. When the AFTER phrase is specified in a WRITE statement for a print file with the FORMAT clause, the printer sets the column on the line defined in the screen form descriptor.

F 16. In [DS], [Win16] and [Linux], the results obtained by executing a WRITE statement with an ADVANCING phrase for a line sequential file are undefined.

In [HP], [Win32], [Winx64], [Solaris], [LinuxIPF], [Linux64] and [.NET], if a WRITE statement with an ADVANCING phrase is executed for a line sequential file, the control character that corresponds to the ADVANCING phrase is output to the file. If an ADVANCING phrase is written, control characters are output to the file as follows:

a. If a positive number is specified in integer-1 or identifier-2, line feed codes that are equivalent to the specified value are output to the file.

b. If a negative number is specified in identifier-2, the result is undefined.

c. If PAGE clause is specified, a form feed code is output to the file.

d. If mnemonic-name-1 is specified, the action taken depends on the function-name associated with mnemonic-name-1, as follows:

- When the function name is SLC or CTL, the action taken is the same as when 0 is specified in integer-1 or identifier-2.
- When the function name is CHANNEL-01, the action taken is the same as when PAGE is written.
- When the function name is any of CHANNEL-02 to CHANNEL-12, the action taken is the same as when 1 is specified in integer-1 or identifier-2.

e. If the BEFORE phrase is specified, record-name-1 record is output to the file before a control code is output according to rules (a) to (d).

f. If the AFTER phrase is specified, a record created by adding a return code to the end of the record-name-1 record is output to the file after a control code is output according to rules (a) to (d).

g. If 0 is specified in integer-1 or identifier-2, a record created by adding a return code to the beginning and end of the record-name-1 record is output to the file, regardless of the presence or absence of the BEFORE or AFTER phrase.

17. To output a control record, the WRITE statement that satisfies the following rules must be executed:

a. In a WRITE statement for a print file having the FORMAT clause, record-name-1 must be the record specified in the CONTROL RECORDS clause.

b. In a WRITE statement for a print file without the FORMAT clause, mnemonic-name-1 must be the mnemonic name corresponding to the function name CTL.

For the format of a control record, refer to the "NetCOBOL User's Guide"

18. A WRITE statement for a control record must be followed by a WRITE statement specifying the AFTER ADVANCING PAGE phrase. A WRITE statement for the control record sets the attribute for the control record as the next page attribute. The page attribute remains unchanged unless the next control record is written.

19. When the CHARACTER TYPE or PRINTING POSITION clause is specified in the record-name-1 record description entry or identifier-1 data description entry, the record is written according to the rules in the following table.

FROM phrase specified or not	C.T or P.P clause specified in record-name-1 record description entry	C.T or P.P clause specified in identifier-1 data description entry	Rules applied
Not specified	Specified	-	The statement follows the C.T and P.P clause written in the record-name-1 record description entry
Specified	Not specified	Specified	The statement follows the C.T And P.P clause written in identifier-1 data description entry
	Specified		
	Specified	Not specified	The statement follows the C.T and P.P clause written in record-name-1 record description entry are ignored.

F

C.T clause: CHARACTER TYPE clause

P.P clause: PRINTING POSITION clause

20. If an exception condition occurred during execution of a WRITE statement for a file without the LINAGE clause or if an exception condition other than a page at end condition occurred during execution of a WRITE statement for a file having the LINAGE clause, the following processing is sequentially executed:

    a. Sets the I-O status of the file related to record-name-1.

    b. Executes the USE AFTER STANDARD EXCEPTION procedure related to the file related to record-name-1 when defined, and then passes control according to the USE statement rules. If no USE AFTER EXCEPTION procedure is defined and the FILE STATUS clause is specified in the file related to record-name-1, the statement passes control to the end of the WRITE statement. If neither USE AFTER EXCEPTION procedure nor FILE STATUS clause is specified, the execution result is undefined.

21. If no exception condition occurred during execution of the WRITE statement, the following processing occurs:

    a. The I-O status of the file related to record-name-1 is set.

    b. The records are written.

    c. When the NOT AT END-OF-PAGE phrase is specified, the procedure passes control to imperative-statement-2. After execution of imperative-statement-2, it passes control to the end of the WRITE statement. However, if imperative-statement-2 explicitly passes control via a procedure branching or conditional statement, the procedure passes control according to the rules of the statement. If the NOT AT END-OF-PAGE phrase is omitted, it passes control to the end of the WRITE statement.

Page at End and Page Overflow Conditions in a File Having the LINAGE Phrase Rules

1. Imperative-statement-1 is executed if print page presentation reaches the logical end during execution of a WRITE statement with the END-OF-PAGE phrase. The logical end is specified in the LINAGE clause in the file description entry related to record-name-1.

2. When the WRITE statement is executed on a file for which the LINAGE clause is specified in the file description entry, a page at end condition or page overflow condition may occur. The page at end condition occurs only when the END-OF-PAGE phrase is specified in the WRITE statement.

    a. The page at end condition occurs during printing or output of a line feed in the footing area of the page body. This is because the LINAGE-COUNTER value becomes equal to or greater than integer-2 or data-name-2 in the FOOTING phrase of the LINAGE clause after the line feed is output by the WRITE statement. At this time, imperative-statement-1 is executed after execution of the WRITE statement.

    b. The page overflow condition occurs if a line feed output by the WRITE statement exceeds the current page body. This is because the LINAGE-COUNTER value exceeds integer-1 or data-name-1 in the LINAGE clause after line feed is output by the WRITE statement. At this time, a record is output to the logical page before

(BEFORE) or after (AFTER) the device is positioned at the first line on the body of the next logical page specified in the LINAGE clause according to the ADVANCING phrase. When the END-OF-PAGE phrase is specified, the record is output, then imperative-statement-1 is executed.

c. A page overflow condition occurs if the LINAGE-COUNTER value exceeds both integer-2 or data-name-2 in the FOOTING phrase of the LINAGE clause and integer-1 or data-name-1 in the LINAGE clause after the line feed is output by the WRITE statement.

F

Rules for WRITE Statements for Print Files Specifying the FORMAT Clause

1. If a form descriptor name is specified in the form descriptor name field of a control record and the WRITE statement is executed for the control record, the next page will be a fixed-format page.
2. When a blank is specified in the form descriptor name field of a control record and the WRITE statement is executed for the control record, the next page will be an undefined-format page. Immediately after a print file having the FORMAT clause is opened, the next page will be assumed to be an undefined-format page.
3. When outputting a chart record, a form descriptor name must be specified in the data item specified in the FORMAT clause. At this time, the chart record is output on a fixed-format page defined with the form descriptor name and edited according to the form descriptor.
4. When outputting a line record, a blank must be specified in the data item specified in the FORMAT clause.
5. For outputting a record to a fixed-format page, a partition name or item group name must be moved to the data item specified in the GROUP clause.
6. If a form descriptor name specified in the FORMAT clause data item differs from the current value when writing a chart record, the WRITE statement to be executed must have the AFTER ADVANCING PAGE phrase.
7. For outputting a chart record at the head of a page, the ADVANCING phrase must be specified together with the AFTER ADVANCING PAGE phrase.
8. After the chart record has been output, the line is assumed to be fed to the last line defined in the form descriptor.
9. When outputting a chart record to a floating partition, an exception condition may occur if the overall chart record could not be output at the output position obtained after the ADVANCING phrase is evaluated.

6.4.55 WRITE Statement (Relative and Indexed Files)

This statement writes records to a file.

Format

WRITE record-name-1

[FROM identifier-1]

[INVALID KEY imperative-statement-1]

[NOT INVALID KEY imperative-statement-2]

[END-WRITE]

Syntax Rules

1. Record-name-1 must be the name of the record defined in the FILE SECTION of the DATA DIVISION. It can be qualified by the file name.
2. When a data item is specified in identifier-1, the same area must not be specified for record-name-1 and identifier-1. A function-identifier specified in identifier-1 must be an alphanumeric function.
3. If no USE AFTER STANDARD EXCEPTION procedure of the file related to record-name-1 is defined, the INVALID KEY phrase must be specified.

F

In this implementation of COBOL, however, both the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.

General Rules

Rules Common to Both Relative and Indexed Files

1. The file related to record-name-1 must have been opened in output, input-output mode, or extended mode before the WRITE statement is executed.
2. After a WRITE statement is executed successfully, the data of record-name-1 is not accessible in the record area unless the file related to record-name-1 is specified in the SAME RECORD AREA clause of the I-O-CONTROL paragraph. In this case, the program can reference record-name-1.
3. A WRITE statement having the FROM phrase produce the same result as the following statements executed in the provided order:
    a. Statements according to the MOVE statement rules

       MOVE identifier-1 TO record-name-1

    b. WRITE statement without FROM phrase
4. After the WRITE statement, the information in identifier-1 area is still usable.
5. The file position indicator remains unchanged after the WRITE statement has been executed. In effect, it remains pointed at the end of the file, although the physical file size has increased by the written record.
6. The value of the I-O status of the file related to record-name-1 is updated after execution of the WRITE statement.
7. The record is passed to the I-O control system after execution of the WRITE statement.
8. Data in record-name-1 must not exceed the maximum record length of the file related to record-name-1. Data in record-name-1 must not be less than the minimum record length of the file related to record-name-1. If these conditions are not satisfied, the WRITE statement will fail and no record will be output. At this time, the record area of the file related to record-name-1 remains unchanged. The I-O status of the file related to record-name-1 is updated with a value indicating the cause of the state.

   F

   Note that the compiler accepts the WRITE successfully when a record smaller than the minimum record size is specified, as it cannot predict the run-time environment.
9. If an invalid key condition occurs during execution of a WRITE statement, the statement may fail. The record area of the file related to record-name-1 remains unchanged. After a value indicating an invalid key condition is moved to the I-O status of the file related to record-name-1, control is passed according to the rules in the topic titled "INVALID KEY Phrase" of the "Common Statement Rules" section earlier in this chapter.
10. If an exception condition occurs other than an invalid key condition during execution of a WRITE statement, the I-O status of the file related to record-name-1 is updated with a value indicating the cause of the failure, and control is passed according to the rules in the topic titled "INVALID KEY Phrase" in the "Common Statement Rules" section earlier in this chapter.
11. If no exception condition occurs during execution of the WRITE statement, the following processing is sequentially performed:
    a. The I-O status of the file related to record-name-1 is set.
    b. The record is written.
    c. Control is passed according to the rules in the topic titled "INVALID KEY phrase" in the "Common Statement Rules" section earlier in this chapter.

F

12. When records are added to a file opened in extended and shared mode with the WRITE statement for the file connector of two or more files, the order of record addition may be undefined.
13. When AUTOMATIC is specified in the LOCK MODE phrase in the file control entry related to record-name-1, the existing record will be unlocked after the WRITE statement has executed successfully.

Rules for Relative Files

1. The file position indicator remains unchanged after execution of the WRITE statement.

2. When a WRITE statement is executed for the sequential access file opened in output mode, it automatically sets the relative record number and writes record-name-1. Relative record numbers are generated from the first record in an ascending order. When the RELATIVE KEY phrase is specified in the ACCESS MODE clause of the file related to record-name-1, the I-O control system moves the relative record number of the record output to the relative key item.
3. When the WRITE statement is executed for a random or dynamic access file opened in output mode, the relative record number of the record to be output must have been moved to the relative key item before the WRITE statement is executed.
4. When a WRITE statement is executed for a file opened in extended mode, the file must be a sequential access file. The WRITE statement adds record-name-1 to the file. The relative record number of the record in the WRITE statement is the maximum relative record number in the file plus 1.

   The relative number is incremented by one when the next record is written. When the RELATIVE KEY phrase is specified in the ACCESS MODE clause of the file related to record-name-1, the relative number of the record that is output is moved to the relative key item.
5. When a WRITE statement is executed for a file opened in input-output mode, the file must be a random or dynamic access file. Before execution of the WRITE statement, the relative record number of the record to be output must have been moved to the relative key item. WRITE inserts record-name-1 into the file.
6. An invalid key condition may occur:
   a. If a record having the same relative key item already exists in a random or dynamic access file when the WRITE statement is executed for the file.
   b. If the statement attempts to write the record over the file segment externally defined explicitly or implicitly.
   c. If the number of significant digits of the relative record number exceeds the size of the relative key item.

Rules for Indexed Files

1. The file position indicator remains unchanged after execution of a WRITE statement.

   F However, the file position indicator is undefined when:

   F a. The DUPLICATES phrase is specified in the RECORD KEY clause of the file related to record-name-1
   b. The file position indicator is set with a START statement with the REVERSED ORDER phrase
2. Before the WRITE statement is executed, a required value must have been moved to the prime record key item.
3. If the DUPLICATES phrase is omitted from the RECORD KEY clause of the file related to record-name-1, the value of the prime record key item must be unique throughout all records in the file.

   F If DUPLICATES is specified, the value may not be unique.
4. The WRITE statement writes data in the record area of record-name-1. The I-O control system uses the value of the record key item to later access the record according to any record key.
5. To output a record to a sequential access file, a value must be moved to the prime record key item according to the rules listed below.
   a. When the file is opened in output mode.

      If the DUPLICATES phrase is omitted in the RECORD KEY clause, a value must be moved to the prime record key item so that values of the prime record keys are in ascending order of the prime record key items.

      F If the DUPLICATES phrase is specified in the RECORD KEY clause, the same value as in the prime record key of the previously written record can be output.
   b. When the file is opened in extended mode.

      If the DUPLICATES phrase is omitted in the RECORD KEY clause, a value greater than the maximum prime record key in the file must be moved to the prime record key item.

F When the DUPLICATES clause is specified in the RECORD KEY clause, the same value as of the maximum prime record key in the file can be output.

6. When writing records to a random or dynamic access file, the records may be output in any sequence.
7. When the DUPLICATES phrase is specified in the ALTERNATE RECORD KEY clause of the file related to record-name-1, the value of the alternate record key item need not be unique throughout all the records in the file. At this time, records are stored so that they can be read with a READ statement having the NEXT phrase in the order they are written.
8. An invalid key exception occurs:
   a. In a WRITE statement for a sequential access file having the RECORD KEY clause omitting the DUPLICATES phrase, if the record key value is equal to or less than that of the previous record.
F b. In a WRITE statement for a sequential access file having the RECORD KEY clause with the DUPLICATES phrase, if the record key value is less than that of the previous record.
   c. In a WRITE statement for a random or dynamic access file having the RECORD KEY clause omitting the DUPLICATES phrase, if the record key value is equal to that of a record in the file.
   d. In a WRITE statement for a random or dynamic access file having the RECORD KEY clause omitting the DUPLICATES phrase, if the alternate record key value is equal to that of a record in the file.
   e. If the statement attempts to write records over the file segment externally defined explicitly or implicitly.

F
6.4.56 WRITE Statement (Presentation File)

This statement writes records to a file.

Format

WRITE record-name-1 [FROM identifier-1]

[END-WRITE]

Syntax Rules

1. Record-name-1 must be the name of the record defined in the FILE SECTION of the DATA DIVISION. It can be qualified by the file name.
2. When a data item is specified in identifier-1, the same area must not be specified for record-name-1 and identifier-1. A function-identifier specified in identifier-1 must be an alphanumeric function.
3. When identifier-1 is a data item, the encoding form of identifier-1 must be same as the encoding form of record-name-1 specified explicitly or implicitly. This rule is specific to [Win32], [Winx64] and [Linux64].

General Rules

1. The file related to record-name-1 must have been opened in output, input-output mode, or extended mode before the WRITE statement is executed.
2. After a WRITE statement is executed successfully, the record of record-name-1 is not accessible in the record area. However, the record remains accessible after execution of the WRITE statement if the file related to record-name-1 is specified in the SAME RECORD AREA clause of the I-O-CONTROL paragraph. In this case, the program can reference record-name-1.
3. A WRITE statement having the FROM phrase produces the same result as the following statements executed in the provided order:
   a. Statement according to the MOVE statement rules

```
MOVE identifier-1 TO record-name-1
```

   b. Same WRITE statement but without FROM phrase

4. After a WRITE statement has been executed, the information of record-name-1 area cannot be used unless the file related to record-name-1 is specified in the SAME RECORD AREA clause. However, information in identifier-1 area is still usable.
5. The value of the I-O status of the file related to record-name-1 is updated after execution of the WRITE statement.
6. The record is passed to the I-O control system after execution of the WRITE statement.

6.5 General Rules for Functions

Functions call a specific process using data as arguments, storing the processing result (function value) in a temporary data item and returning it to the program.

6.5.1 Function Call Format

The function name must be entered in the format shown below. The function name is called a function-identifier.

The function name is the name of a function. A function-name is a COBOL word.

The argument is a value used to determine the function value. For some functions, the range of arguments can be limited. The number, sequence, and value of arguments depend on each of functions.

Function-identifiers can only be written in the PROCEDURE DIVISION.

Format

FUNCTION function-name-1 [({argument-1}...)] [reference-modifier]

6.5.2 Types of Arguments

Function arguments can be an identifier, arithmetic expression, or literal. The data that can be specified in function arguments is determined by the type of argument that the function requires.

There are five types of arguments: alphabetic, alphanumeric, numeric character, integer and national character. Arguments must be specified according to the rules listed below.

1. If the argument type is alphabetic, the argument must be an alphabetic class item or a nonnumeric literal containing alphabetic characters only. The argument size can also be used to determine the function value.
2. If the argument type is alphanumeric, the argument must be an alphabetic data item, alphanumeric data item, alphanumeric edited data item, or nonnumeric literal. The argument size can also be used to determine the function value.
3. If the argument type is numeric, the argument must be an arithmetic expression. The value of the arithmetic expression (including sign) is used to determine the function value.
4. If the argument type is integer, the argument must be an arithmetic expression that has an integer value. The value of the arithmetic expression (including sign) is used to determine the function value.
5. If the argument type is national, an elementary national data item must be specified. The argument size may be used to determine a function value.

F

Some functions may use other data items or literals.

The range of argument values depends on each of the functions. If a value outside the defined range is specified in an argument, the function value may be undefined.

6.5.3 Rules Applied for Specifying a Table as an Argument

If you want to specify a function argument that is composed of all the table elements of a dimension, specify the data-name with the keyword ALL as a subscript, instead of writing individual subscripted data-names. Specifying ALL as a subscript passes all the table elements of the dimension to the function as if they were specified from left to right in the order of occurrence.

See the table example defined below. In this example, (a) and (b) as well as (c) and (d) will have the same value.

```
01  TBL.
  02  A OCCURS 3.
    03  B OCCURS 4.
      04  C PIC X.
```

(a) FUNCTION MAX (A(1) A(2) A(3))

(b) FUNCTION MAX (A(ALL))

(c) FUNCTION MAX (C(1 1) C(1 2) C(1 3) C(1 4))

(d) FUNCTION MAX (C(1 ALL))

When the subscript ALL is for a table dimension that specifies the OCCURS clause with the DEPENDING ON phrase, the upper limit of ALL follows the data-name having the DEPENDING ON phrase.

After evaluation of the subscript ALL, at least one table element must be referenced. Otherwise, no function value is returned.

6.5.4 Function Types

There are five function types:

- alphanumeric
- numeric
- integer
- F pointer-data
- national

These functions are called, depending on the function type,

- "alphanumeric function"
- "numeric function"
- "integer function"
- F "pointer-data function"
- "national function

Alphanumeric Function

The category and class of the temporary data item for an alphanumeric function are alphanumeric. The data item type is USAGE DISPLAY. An alphanumeric function-identifier may be used as the source, or sending, item for alphanumeric data items only. The size of the alphanumeric function value depends on each function.

Numeric Function

The category and class of the temporary data item for a numeric function is numeric. The data item has a sign. The numeric function-identifier can be used only as a source in arithmetic expressions. Note that it cannot be used with in integer receiving fields. A numeric function can be used wherever a numeric operand is allowed in the general format of a statement. The attribute and precision of the numeric function value depends on each function.

Integer Function

The category and class of the temporary data item for an integer function is numeric. The data item has a sign. The integer function-identifier can be a source for integer items only in arithmetic expressions. An integer function can be used wherever an integer operand is allowed in the general format of a statement. The attribute and precision of the integer function value depends on each function.

F

Pointer Data Function

The temporary data item for a pointer data function is used for pointer data. The pointer data function-identifier can be used as a source for pointer data items only.

F

National Function

The class and category of a national function are national. The function-identifier of a national function can only be used in a place where a source or sending national data item can be used in the PROCEDURE DIVISION. The number of character positions for the function value of a national function depends on the function type.

6.6 Functions

This section describes each function. For details on functions specific to [.NET], refer to "Functions" in "Chapter 12 Microsoft .NET Support".

6.6.1 ACOS Function

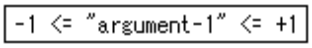
This function returns the arc cosine of argument-1.

Format

FUNCTION ACOS (argument-1)

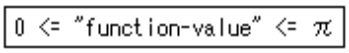
Argument

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be within the following range:



Function Value

1. The function value is an approximate value of the arc cosine of argument-1. The unit is radians.
2. The range of the function value must be:



Function Type

The function type is numeric.

F

6.6.2 ADDR Function

This function returns the address of argument-1.

For details about how to use the examples, refer to "ADDR and LENG functions" in the "Syntax Samples".

Format

FUNCTION ADDR (argument-1)

Argument

Argument-1 must not be an internal Boolean data item.

Function Value

The function value is the address of argument-1.

Function Type

The function type is pointer data.

ADDR Function Specific Rules

The ADDR function can only be specified in the following places:

- Pointer qualifier
- DISPLAY statement
- Sending side of a MOVE statement
- Relation condition in the IF and EVALUATE statements

If the ADDR function is specified in the DISPLAY statement, the address of argument-1 is displayed in hexadecimal notation.

6.6.3 ANNUITY Function

This function returns a uniform payment when argument-1 is the interest rate, argument-2 is the period of uniform payment, and the capital is 1.

Format

FUNCTION ANNUITY (argument-1 argument-2)

Arguments

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be zero or greater.
3. The type of argument-2 must be integer.
4. The value of argument-2 must be a positive integer.

Function Value

The function value is:

- When argument-1 is zero, it is an approximate value of the following arithmetic expression:

```
1/"argument-2"
```

- When argument-1 is not zero, it is an approximate value of the following arithmetic expression:

```
"argument-1"/(1 - (1 + "argument-1" ** (-"argument-2"))
```

Function Type

The function type is numeric.

6.6.4 ASIN Function

This function returns the arc sine of argument-1.

Format

FUNCTION ASIN (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be within the following range:

```
-1 <= "argument-1" <= +1
```

Function Value

1. The function value is the arc sin of argument-1. The unit is radians.
2. The range of the function value must be:

```
-π/2 <= "function-value" <= +π/2
```

Function Type

The function type is numeric.

6.6.5 ATAN Function

This function returns the arc tangent of argument-1.

Format

FUNCTION ATAN (argument-1)

Argument

The type of argument-1 must be numeric.

Function Value

1. The function value is an approximate value of the arc tangent of argument-1. The unit is radians.
2. The range of the function value must be:

```
-π/2 < "function-value" < +π/2
```

Function Type

The function type is numeric.

F

6.6.6 CAST-ALPHANUMERIC Function

This function returns the national or national edited character-string in argument-1 as an alphanumeric character-string.

Format

FUNCTION CAST-ALPHANUMERIC (argument-1)

Argument

The length of argument-1 must be at least one character, and its class must be national.

Function Value

1. The function value is the same character-string as argument-1 because each national character is regarded as an alphanumeric character.
2. The number of characters in the function value is double the number of characters in argument-1.

Function Type

The function type is alphanumeric.

Rules specific to CAST-ALPHANUMERIC function

This function cannot be used when the operation mode is Unicode (ISO/IEC 10646_1).

6.6.7 CHAR Function

This function returns the character at argument-1 position in the collating sequence.

Format

FUNCTION CHAR (argument-1)

Argument

1. The type of argument-1 must be integer.
2. Assume n as the maximum precedence value. The value of argument-1 must be from 1 to n.

Function Value

1. The function value is the character of argument-1 position in the collating sequence. The collating sequence is defined in the ALPHABET clause in the SPECIAL-NAMES paragraph. If the ALPHABET clause is omitted, the function value is determined according to the collating sequence of the computer character set.
2. If two or more characters are found at argument-1 position in the collating sequence, the first literal character specified in the sequence of the ALPHABET clause is returned as the function value.
3. The encoding form of the function value is default encoding form.

Function Type

The function type is alphanumeric.

6.6.8 COS Function

This function returns the cosine of argument-1.

For details about how to use the examples, refer to "SIN, COS and TAN functions" in the "Syntax Samples".

Format

FUNCTION COS (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The unit of argument-1 must be radians.

Function Value

1. The function value is an approximate value of the cosine of argument-1.

2. The range of the function value must be:

```
-1 <= "function-value" <= +1
```

Function Type

The function type is numeric.

6.6.9 CURRENT-DATE Function

This function returns the current date and time.

For details about how to use the examples, refer to "CURRENT-DATE function" in the "Syntax Samples".

Format

FUNCTION CURRENT-DATE

Function Value

The function value is 21 alphanumeric characters indicating the date and time when the function is executed as well as the time difference from the standard time. The meaning of each character in the function value is listed below.

Character Position	Value
1 to 4	4-digit number indicating the year.
5 to 6	2-digit number indicating the month. The value ranges from 01 to 12.
7 to 8	2-digit number indicating the day. The value ranges from 01 to 31.
9 to 10	2-digit number indicating hours. The value ranges from 00 to 23.
11 to 12	2-digit number indicating minutes. The value ranges from 00 to 59.
13 to 14	2-digit number indicating seconds. The value ranges from 00 to 59.
15 to 16	2-digit number indicating 1/100 seconds. The value ranges from 00 to 99. (*1)
17	Sign indicating whether the local time (time indicated in character positions 1 to 16) is ahead or behind the Greenwich mean time. A minus (-) sign indicates that the local time is behind Greenwich mean time, and a plus (+) sign indicates that it is ahead. (*2)
18 to 19	2-digit number indicating the number of hours ahead or behind Greenwich mean time. The value ranges from 00 to 12 when character position 17 is "-"; and ranges from 00 to 13 when it is "+." (*2)
20 to 21	2-digit number indicating the number of minutes ahead or behind Greenwich mean time. The value ranges from 00 to 59. (*2)

*1 Character positions 15 and 16 are set to 00 if the system cannot read 1/100 seconds.

*2 Character positions 17 to 21 are set to 00000 if the system cannot read the time difference between the local time and Greenwich mean time.

Function Type

The function type is alphanumeric.

6.6.10 DATE-OF-INTEGER Function

This function converts and returns argument-1 to the standard format date.

For details about how to use the examples, refer to "INTEGER-OF-DATE and DATE-OF-INTEGER Functions" in the "Syntax Samples".

Format

FUNCTION DATE-OF-INTEGER (argument-1)

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be a positive integer indicating a serial day starting from January 1, 1601.

Function Value

1. The function value is the standard format date equivalent to the serial day specified in argument-1.
2. The function value is in the format of yyyymmdd; where yyyy is the year, mm is the month, and dd is the day.

Function Type

The function type is integer.

6.6.11 DAY-OF-INTEGER Function

This function converts and returns argument-1 to the year and day format.

Format

FUNCTION DAY-OF-INTEGER (argument-1)

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be a positive integer indicating a serial day starting from January 1, 1601.

Function Value

1. The function value is the year and day format date equivalent to the serial day specified in argument-1.
2. The function value is set in the format of yyyyddd; where yyyy is the year and ddd is the serial day in the year.

Function Type

The function type is integer.

F

6.6.12 DISPLAY-OF Function

This function replaces national characters in argument-1 with alphanumeric characters. This function is available for [Win32], [Winx64], [Solaris], [Linux64] and [.NET].

Format

FUNCTION DISPLAY-OF (argument-1 [argument-2])

Argument

1. The class of argument-1 must be national character.
2. The class of argument-2 must be an alphabetic or alphanumeric character, and should be one character in length.
3. When argument-2 is a data item, its encoding form must be default encoding form. This rule is specific to [Win32], [Winx64] and [Linux64].

4. Argument-2 specifies the alphanumeric character to use when there is no alphanumeric character that corresponds to the argument-1 national character.

Function Value

1. Function value is a character string that replaces national characters included in argument-1 with the corresponding alphanumeric characters.
2. When argument-2 is specified, it is used when national characters in argument-1 do not have corresponding alphanumeric characters.
3. An execution time exception is generated when no argument-2 is specified and there are no alphanumeric characters that correspond to national characters in argument-1.
4. The function value must be large enough to maintain the character string after it is converted.
5. Character strings in argument-1 that contain trailing blanks are converted without the trailing blanks.
6. The encoding form of the function value is default encoding form.

Function Type

The function type is alphanumeric.

DISPLAY-OF Function Specific Rules

This function can be used only when the operation mode is Unicode (ISO/IEC 10646-1).

6.6.13 FACTORIAL Function

This function returns the factorial of argument-1.

Format

FUNCTION FACTORIAL (argument-1)

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be zero or a positive integer.

Function Value

The function value is:

- When argument-1 is zero, it is 1.
- When argument-1 is a positive numeric value, it is the factorial of argument-1.

Function Type

The function type is integer.

6.6.14 INTEGER Function

This function returns the maximum integer equal to or less than argument-1.

Format

FUNCTION INTEGER (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The floating-point cannot be specified for an argument.

Function Value

The function value is the maximum integer equal to or less than argument-1. For example, when the value of argument-1 is +1.5, the function value is +1. When argument-1 is -1.5, the function value is -2.

Function Type

The function type is integer.

6.6.15 INTEGER-OF-DATE Function

This function converts and returns argument-1 (standard format date) to a serial day.

For details about how to use the examples, refer to "INTEGER-OF-DATE and DATE-OF-INTEGER Functions" in the "Syntax Samples".

Format

FUNCTION INTEGER-OF-DATE (argument-1)

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be an eight-digit integer in the format of yyyymmdd.
3. Argument-1 must be in the standard format using the following arithmetic expression.

```
(yyyy * 10000) + (mm * 100) + dd
```

The values yyyy, mm, and dd must follow the following rules:

a. yyyy is the numeric value indicating the year. It must be greater than 1600.

b. mm is the numeric value indicating the month. It must be in the range from 1 to 12.

c. dd is the numeric value indicating the day. It must be in the range from 1 to 31.

d. A combination of yyyy, mm, and dd must be a value indicating a proper date.

Function Value

The function value is an integer indicating a serial day equivalent to argument-1 (standard format date). The serial day begins from January 1, 1601.

Function Type

The function type is integer.

6.6.16 INTEGER-OF-DAY Function

This function converts and returns argument-1 (date format) to a serial day.

Format

FUNCTION INTEGER-OF-DAY (argument-1)

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be a seven-digit integer in the format of yyyyddd.
3. argument-1 must be in the year and day format using the following arithmetic expression.

```
(yyyy * 1000) + ddd
```

The values yyyy and ddd must follow the following rules:

a. yyyy is the numeric value indicating the year. It must be greater than 1600.

b. ddd is the numeric value indicating the serial day in the year. It must be in the range from 1 to 366.

c. A combination of yyyy and ddd must be a value indicating a proper date.

Function Value

The function value is an integer indicating a serial day equivalent to argument-1 (year and day format date). The serial day begins from January 1, 1601.

Function Type

The function type is integer.

6.6.17 INTEGER-PART Function

This function returns the integer part of argument-1.

Format

FUNCTION INTEGER-PART (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The floating-point cannot be specified for an argument.

Function Value

The function value is:

1. When the value of argument-1 is 0, it is 0.
2. When the value of argument-1 is a positive value, it is the maximum integer equal to or less than argument-1. For example, when argument-1 is +1.5, the function value is +1.
3. When the value of argument-1 is a negative value, it is the minimum integer equal to or greater than argument-1. For example, when argument-1 is -1.5, the function value is -1.

Function Type

The function type is integer.

F

6.6.18 LENG Function

This function returns the length (number of bytes or number of bits) of argument-1.

For details about how to use the examples, refer to "ADDR and LENG Functions" in the "Syntax Samples".

Format

FUNCTION LENG (argument-1)

Argument

The type of argument-1 must be a data item, nonnumeric literal, hexadecimal nonnumeric literal or national language nonnumeric literal.

The national nonnumeric literal cannot be specified for argument-1 in [Linux].

Function Value

The function value is the size (number of bytes) of argument-1. However, the function value is the size (number of bits) of argument-1 in the case of an internal Boolean function.

The size of a national nonnumeric literal depends on the default encoding form.

Function Type

The function type is integer.

LENG Function Specific Rules

The LENG function can be specified only in the following places:

- Source data item of MOVE statement
- Arithmetic expression
- Integer numeric literal items in the PROCEDURE DIVISION

6.6.19 LENGTH Function

This function returns the length (number of character positions or national character positions(*)) of argument-1

Format

FUNCTION LENGTH (argument-1)

Argument

1. The type of argument-1 must be a nonnumeric literal, hexadecimal nonnumeric literal(*), national literal(*), or any data item
2. If the OCCURS clause having the DEPENDING ON phrase is specified in a data item subordinate to argument-1, the function value is determined using the data-name having the DEPENDING ON phrase at evaluation of the LENGTH function.

F

* : Extended functions or functions specific to NetCOBOL.

- national character positions
- hexadecimal nonnumeric literal
- national literal

Function Value

1. The function value is listed in the table below.

Argument-1	Function Value
Nonnumeric literal	Number of character positions of argument-1 (number of bytes)
Hexadecimal nonnumeric literal	

Argument-1	Function Value
National literal	Number of national character positions of argument-1 (number of national characters)
Alphabetic	Number of character positions of argument-1 (number of bytes)
Alphanumeric	
Alphanumeric edited	
Numeric edited data items	
Numeric	Number of bytes in the storage area used by argument-1
Group	
Index	
Pointer-data item	
National and national edited data items	Number of national character positions of argument-1 (number of national characters)
External Boolean data item	Number of Boolean character positions of argument-1 (number of bytes)
Internal Boolean data item	Number of bits of argument-1

F

* : Extended functions or functions specific to NetCOBOL.

2. When argument-1 is a group item including variable occurrence data items, the function value is determined by evaluating the data-name having the DEPENDING ON phrase in the OCCURS clause in the variable occurrence data item. Evaluation follows the rules for when the OCCURS clause is specified in the source data item.
3. If an implicit FILLER item exists, the function value contains the number of characters of the items.

Function Type

The function type is integer.

6.6.20 LOG Function

This function returns the log of argument-1 using e as the base.

Format

FUNCTION LOG (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be positive.

Function Value

The function value is an approximate value of the natural logarithm of argument-1.

Function Type

The function type is numeric.

6.6.21 LOG10 Function

This function returns the log of argument-1 using 10 as the base.

Format

FUNCTION LOG10 (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be positive.

Function Value

The function value is an approximate value of the common logarithm of argument-1.

Function Type

The function type is numeric.

6.6.22 LOWER-CASE Function

This function returns lowercase alphabetic characters equivalent to uppercase alphabetic characters in argument-1.

For details about how to use the examples, refer to "LOWER-CASE and UPPER-CASE Functions" in the "Syntax Samples".

Format

FUNCTION LOWER-CASE (argument-1)

Argument

1. The type of argument-1 must be alphabetic or alphanumeric.
2. The length of argument-1 must be 1 character or more.

Function Value

1. The function value is a character string of lowercase alphabetic characters that have been converted from uppercase alphabetic characters. In other words, the result is the same as argument-1 except that the uppercase alphabetic characters have been replaced.
2. The length of the function value is the same as of argument-1.
3. The encoding form of the function value is the same as argument-1.

Function Type

The function type is alphanumeric.

6.6.23 MAX Function

This function returns the maximum value of argument-1.

For details about how to use the examples, refer to "MAX and MIN Functions" in the "Syntax Samples".

Format

FUNCTION MAX ({argument-1}...)

Argument

1. Argument-1 can be of type alphabetic, alphanumeric, integer, or numeric .

2. All argument-1 elements must be of matching types as shown in the function type table below.

Function Value

1. The function value is the maximum value in the list of argument-1.
2. If two or more values in argument-1 have the same maximum value, the high order end is returned as function value.
3. Comparison for determining the maximum value follows the rules for relation conditions.
4. When the function type is alphanumeric, the length of the function value is the same as of argument-1 having the maximum value.

Function Types

The table below lists the function types.

Type of argument-1	Function Type
All alphabetic or alphanumeric characters	Alphanumeric character
All integers	Integer
All numeric or mix of numeric characters and integers	Numeric character

6.6.24 MEAN Function

This function returns the arithmetic mean value of argument-1.

Format

FUNCTION MEAN ({argument-1}...)

Argument

The type of argument-1 must be numeric.

Function Value

The function value is the arithmetic mean value of the list of argument-1. In other words, the value will be the sum of the list of argument-1 elements divided by the number of elements in argument-1.

Function Type

The function type is numeric.

6.6.25 MEDIAN Function

This function returns the median of argument-1.

Format

FUNCTION MEDIAN ({argument-1}...)

Argument

The type of argument-1 must be numeric.

Function Value

1. The function value is the median when the list of argument-1 is sorted in ascending order. The median is determined as follows:
    a. When the number of elements of argument-1 is odd-numbered, the middle value after sorting is returned.
    b. When the number of elements of argument-1 is even-numbered, the mean value of the two middle values after sorting is returned.
2. Comparison for sorting must follow the relation condition rules.

Function Type

The function type is numeric.

6.6.26 MIDRANGE Function

This function returns the arithmetic mean value of the minimum and maximum values of argument-1.

Format

FUNCTION MIDRANGE ({argument-1}...)

Argument

The type of argument-1 must be numeric.

Function Value

The function value is the arithmetic mean value of the maximum and minimum values in the list of argument-1. In other words, the value will be the sum of the minimum and maximum values divided by two. Comparison for determining the maximum and minimum values follows the relational condition rules.

Function Type

The function type is numeric.

6.6.27 MIN Function

This function returns the minimum value of argument-1.

For details about how to use the examples, refer to "MAX and MIN Functions" in the "Syntax Samples".

Format

FUNCTION MIN ({argument-1}...)

Argument

1. Argument-1 can be of type alphabetic, alphanumeric, integer, or numeric.
2. All argument-1 elements must be of matching types as shown in the function type table below.

Function Value

1. The function value is the minimum argument-1 in the list of argument-1 elements.
2. If more than one argument-1 has the same minimum value, the high order end argument is returned.
3. Comparison for determining the minimum value follows the relation condition rules.
4. When the function type is alphanumeric, the length of the function value is the same as that of argument-1 having the minimum value.

Function Type

The table below lists the function types.

Type of argument-1	Function Type
All alphabetic or alphanumeric characters	Alphanumeric character
All integer	Integer
All numeric or mix of numeric characters and integers	Numeric character

6.6.28 MOD Function

This function returns the integer value of argument-1 using argument-2 as modulus.

Format

FUNCTION MOD (argument-1 argument-2)

Argument

1. The type of argument-1 and argument-2 must be integer.
2. The value of argument-2 must not be zero.

Function Value

1. The function value is an integer value using argument-2 as modulus. It is determined using the following arithmetic expression:

```
"argument-1" - ("argument-2" * FUNCTION INTEGER ("argument-1"/"argument-2"))
```

2. Some function value examples are shown below.

argument-1	argument-2	Function Value
11	5	1
-11	5	4
11	-5	-4
-11	-5	-1

Function Type

The function type is integer.

F

6.6.29 NATIONAL Function

This function replaces the COBOL character set characters in argument-1 with national characters.

Format

FUNCTION NATIONAL (argument-1)

Argument

The length of argument-1 must be at least one character, and its class must be alphabetic, alphanumeric, or numeric. Note that only a zoned decimal data item is accepted when the class is numeric.

Function Value

1. The function value is the same character-string as argument-1, except that the COBOL character set characters are replaced by the corresponding national characters.
2. The number of characters in the function value is the same as the number of characters in argument-1.
3. If a signed zoned decimal data item is specified in argument-1, the sign is ignored.
4. If an invalid character code is specified in argument-1, the result is undefined.
5. The encoding form of the function value is default encoding form.

Function Type

The function type is national.

F

6.6.30 NATIONAL-OF Function

This function replaces alphanumeric characters in argument-1 with national characters. This function is available for [Win32], [Winx64], [Solaris], [Linux64] and [.NET].

Format

FUNCTION NATIONAL-OF (argument-1 [argument-2])

Argument

1. The class of argument-1 must be alphabetic or alphanumeric.
2. Argument-2 must be one character in length, and it must be a national character.
3. The encoding form must be default encoding form when argument-2 is a data item.
4. Argument-2 specifies the national character to use when there is no national character that corresponds to the argument-1 alphanumeric character.

Function Value

1. The function value is a character string that replaces alphanumeric characters in argument-1 with corresponding national characters.
2. When argument-2 is specified, it is used when alphanumeric characters in argument-1 do not have corresponding national characters.
3. An execution time exception is generated when no argument-2 is specified and there are no national characters that correspond to the alphanumeric characters in argument-1.
4. The function value must be large enough to maintain the national character string after it is converted.
5. Character strings in argument-1 that contain trailing blanks are converted without the trailing blanks.
6. The encoding form of the function value is the default encoding form of national data items.

Function Type

The function type is national.

NATIONAL-OF Function Specific Rules

This function can be used only when the default encoding form is Unicode (ISO/IEC 10646-1).

6.6.31 NUMVAL Function

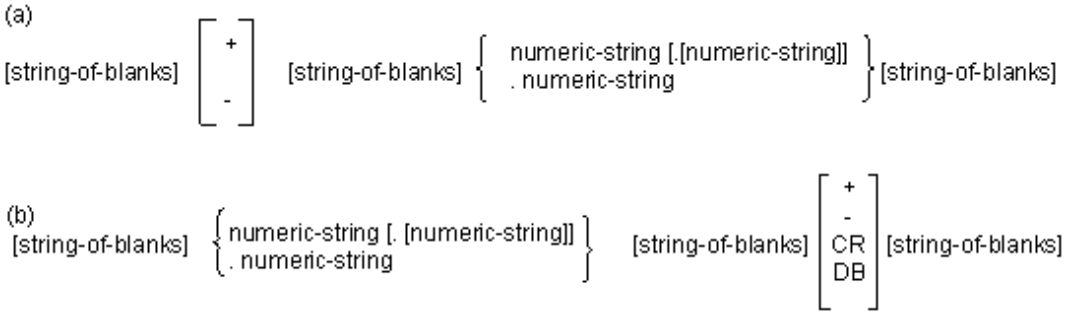
This function converts argument-1 (character string in numeric literal format) to a numeric value and returns it.

Format

FUNCTION NUMVAL (argument-1)

Argument

1. Argument-1 must be a nonnumeric literal or alphanumeric data item. The value of argument-1 must be in either of the formats shown below. The string-of-blanks indicates a sequence of one or more blanks, and the numeric-string indicates a sequence of one or more numeric values.



2. The sum of digits of argument-1, that is, the number of digits in the numeric strings must be 18 or less.
3. When the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma must be entered instead of the period (.) to indicate the decimal point.

Function Value

The function value is the value of argument-1.

Function Type

The function type is numeric.

6.6.32 NUMVAL-C Function

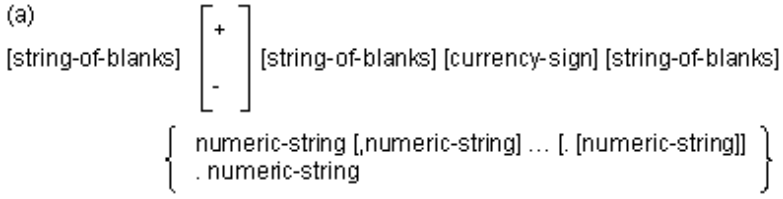
This function converts argument-2 (currency symbol) or the character string in argument-1 containing a comma to a numeric value and returns it.

Format

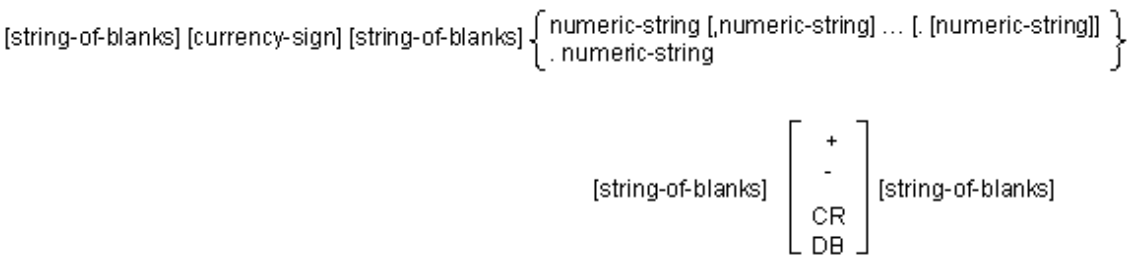
FUNCTION NUMVAL-C (argument-1 [argument-2])

Argument

1. Argument-1 must be a nonnumeric literal or alphanumeric data item. The value of argument-1 must be either of the formats shown below. The string-of-blanks indicates a sequence of one or more blanks, and numeric-string indicates a sequence of one or more numeric values. A currency sign indicates a character string (more than one character) identical to argument-2.



(b)



2. The sum of digits of argument-1; that is, the number of digits in the numeric strings must be 18 or less.
3. When the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma must be entered instead of the period (.) to indicate the decimal point.
4. Argument-2 must be a nonnumeric literal or alphanumeric data item.
5. Argument-2 must specify a currency sign. If argument-2 is omitted, the currency sign specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph is assumed. If the CURRENCY SIGN clause is omitted, the currency sign in the COBOL character set is assumed.

Function Value

The function value is the numeric value of argument-1.

Function Type

The function type is numeric.

6.6.33 ORD Function

This function returns the order (position) of argument-1 in the collating sequence.

Format

FUNCTION ORD (argument-1)

Argument

1. The type of argument-1 must be alphabetic or alphanumeric.
2. The length of argument-1 must be one character.

Function Value

The function value is the order of argument-1 in the collating sequence. The collating sequence is defined in the ALPHABET clause in the SPECIAL-NAMES paragraph. If the ALPHABET clause is omitted, the function value is determined according to the collating sequence of the computer's character set.

Function Type

The function type is integer.

6.6.34 ORD-MAX Function

This function returns the order (position) of the element within argument-1 which has the maximum value.

Format

FUNCTION ORD-MAX ({argument-1}...)

Argument

1. Argument-1 can be of type alphabetic, alphanumeric, integer, or numeric
2. The combination of argument-1 types must be one of the following:
   - All alphabetic characters
   - All alphanumeric characters
   - All integers
   - All numeric characters
   - Mix of numeric characters and integers

Function Value

1. The function value is the position of argument-1 having the maximum value in the list of argument-1 elements. For example, when the second argument-1 from the left has the maximum value in the list of argument-1 elements, 2 is returned.
2. If more than one argument-1 has the same maximum value, the one written furthest to the left is the maximum value. Accordingly, the function returns the value of that position.
3. Comparison for determining the maximum value follows the relational condition rules.

Function Type

The function type is integer.

6.6.35 ORD-MIN Function

This function returns the position of the element of argument-1 which has the minimum value.

Format

FUNCTION ORD-MIN ({argument-1}...)

Argument

1. Argument-1 can be of type alphabetic, alphanumeric, integer, or numeric
2. The combination of argument-1 types must be one of the following:
   - All alphabetic characters
   - All alphanumeric characters
   - All integers
   - All numeric characters
   - Mix of numeric characters and integers

Function Value

1. The function value is the position of argument-1 having the minimum value in the list of argument-1 elements. For example, when the second argument-1 from the left has the minimum value in the list of argument-1 elements, 2 is returned.
2. If more than one argument-1 has the same minimum value, the one written furthest to the left is the minimum value.
3. Comparison for determining the minimum value follows the relational condition rules.

Function Type

The function type is integer.

6.6.36 PRESENT-VALUE Function

This function returns the present value at the end of each term in the list of argument-2 using argument-1 as depreciation rate.

Format

FUNCTION PRESENT-VALUE (argument-1 {argument-2} ...)

Arguments

1. The type of argument-1 and argument-2 must be numeric.
2. Argument-1 must be greater than -1.

Function Value

The function value is an approximate value of the sum of numeric strings calculated from a series of terms as shown below. One term corresponds to each value in argument-2. Exponent n begins from 1 and is incremented by one for each term.

```
"argument-2"/(1 + "argument-1") ** n
```

Function Type

The function type is numeric.

6.6.37 RANDOM Function

This function returns a pseudo random value from uniform distributions.

Format

FUNCTION RANDOM [(argument-1)]

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be zero or a positive integer. It is used to generate a pseudo-random string.
3. The RANDOM function with argument-1 will generate a pseudo-random string using argument-1 as source.
4. If argument-1 is omitted in the RANDOM function first executed in the run unit, a pseudo-random string is generated using zero as source.
5. When determining a random value from the same pseudo random string after execution of the RANDOM function, argument-1 may be omitted. The value of argument-1 is valid until the next RANDOM function having argument-1 appears.

Function Value

1. The function value is any value in the current pseudo-random string.
2. The range of the function value is:

```
0 <= "function value" < 1
```

3. If the value of argument-1 (source value) is the same as in a previous execution, the same pseudo-random string is used.

Function Type

The function type is numeric.

6.6.38 RANGE Function

This function returns the maximum value minus the minimum value of an argument_1.

Format

FUNCTION RANGE ({argument-1}...)

Argument

The type of argument-1 must be numeric or integer.

Function Value

1. The function value is the maximum value minus the minimum value of the list of argument-1.
2. Comparison for determining the maximum and minimum values follows the relational condition rules.

Function Type

The table below lists the function types:

Type of argument-1	Function Type
All integers	Integer
All numeric or mix of numeric characters and integers	Numeric character

6.6.39 REM Function

This function returns the remainder of argument-1 divided by argument-2.

For details about how to use the examples, refer to "REM Function" in the "Syntax Samples".

Format

FUNCTION REM (argument-1 argument-2)

Arguments

1. The type of argument-1 and argument-2 must be numeric.
2. The value of argument-2 must not be zero.
3. A floating-point value cannot be specified as an argument.

Function Value

The function value is the remainder of argument-1 divided by argument-2. It is determined using the following arithmetic expression:

```
"argument-1" - ("argument-2" * FUNCTION INTEGER-PART ("argument-1"/"argument-2"))
```

Function Type

The function type is numeric.

6.6.40 REVERSE Function

This function returns the reverse of the character string in argument-1.

For details about how to use the examples, refer to "REVERSE Function" in the "Syntax Samples".

Format

FUNCTION REVERSE (argument-1)

Argument

1. The type of argument-1 must be numeric or alphanumeric.
2. The length of argument-1 must be one character or more.

Function Value

1. The function value is the reverse of the character string in argument-1. Its length is the same as of argument-1.
2. Assume the length of the character string of argument-1 to be n. The character in the jth (1 <= j <= n) in argument-1 is returned as the (n - j + 1)th character in the function value.
3. The encoding form of the function value is the same as argument-1.

Function Type

The function type is alphanumeric.

6.6.41 SIN Function

This function returns the sine of argument-1.

For details about how to use the examples, refer to "SIN, COS and TAN functions" in the "Syntax Samples".

Format

FUNCTION SIN (argument-1)

Arguments

1. The type of argument-1 must be numeric.
2. The unit of argument-1 must be radian.

Function Value

1. The function value is an approximate value of the sine of argument-1.
2. The range of the function value is:

```
-1 <= "function value" <= +1
```

Function Type

The function type is numeric.

6.6.42 SQRT Function

This function returns the square root of argument-1.

Format

FUNCTION SQRT (argument-1)

Arguments

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be zero or greater.

Function Value

The function value is the absolute value of the approximate value of the square root of argument-1.

Function Type

The function type is numeric.

6.6.43 STANDARD-DEVIATION Function

This function returns the approximate value of the standard deviation of an argument.

Format

FUNCTION STANDARD-DEVIATION ({argument-1} ...)

Argument

The type of argument-1 must be numeric.

Function Value

1. The function value is the approximate value of standard deviation of the list of argument-1 elements.
2. It is determined using the procedures below.
    a. The difference between the mean of argument-1 elements and each argument-1 value is determined and each difference is squared.
    b. All values determined in (a) are added, then divided by the number of argument-1 elements.
    c. The square root of the quotient found in (b) is determined. The function value will be the absolute value of the quotient.
3. If all argument-1 elements have the same value, the function value is zero.

Function Type

The function type is numeric.

F

6.6.44 STORED-CHAR-LENGTH Function

This function returns the length of the effective characters (without trailing blanks) contained in argument-1.

The STORED-CHAR-LENGTH function is available for [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

For details about how to use the examples, refer to "STORED-CHAR-LENG Function" in the "Syntax Samples".

Format

FUNCTION STORED-CHAR-LENGTH (argument-1)

Argument

1. Argument-1 must be alphanumeric or national characters. It must not be a group item.
2. Argument-1 must consist of one or more characters.

Function Value

1. The function value is the number of character positions of the effective characters contained in argument-1.
2. When argument-1 is the alphanumeric class, the function value indicates the number of alphanumeric character positions of the effective characters.
3. When argument-1 is the national class, the function value indicates the number of national character positions of the effective characters.
4. The characters stored in argument-1 must follow the encoding forms dictated by the class of argument-1. Otherwise, the result is undefined.

Function Type

The function type is integer.

6.6.45 SUM Function

This function returns the sum of the elements of argument-1.

For details about how to use the examples, refer to "SUM Function" in the "Syntax Samples".

Format

FUNCTION SUM ({argument-1}...)

Argument

The type of argument-1 must be numeric or integer.

Function Value

The function value is the sum of the list of argument-1 elements.

Function Type

The table below lists the function types.

Type of argument-1	Function Type
All integers	Integer
All numeric or mix of numeric characters and integers	Numeric character

6.6.46 TAN Function

This function returns the tangent of argument-1.

For details about how to use the examples, refer to "SIN, COS and TAN functions" in the "Syntax Samples".

Format

FUNCTION TAN (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The unit of argument-1 must be radian.

Function Value

The function value is the approximate value of the tangent of argument-1.

Function Type

The function type is numeric.

F

6.6.47 UCS2-OF Function

This function converts the encoding forms of the characters included in argument-1 into UCS-2. The UCS2-OF function is available for [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

Format

FUNCTION UCS2-OF (argument-1)

Argument

1. Argument-1 must be alphabetic or alphanumeric.
   - Its encoding form must be UTF-8.
   - It must not be a group item.
2. Argument-1 must consist of one or more characters.

Function Value

1. For the function value, the function converts the character indicated by argument-1 into the corresponding character string represented by UCS-2.
2. The size (bytes) of the function value follows the size of the converted character string.
3. The character in argument-1 must be those that can be stored in an alphanumeric data item. If another type of character is included, the result is undefined.
4. The encoding form of the function value is UTF-16 (default endian).

Function Type

The function type is national.

Rules specific to UCS2-OF function

This function can be used only when the operation mode is Unicode (ISO/IEC 10646-1).

6.6.48 UPPER-CASE Function

This function returns uppercase alphabetic characters equivalent to lowercase alphabetic characters in argument-1.

For details about how to use the examples, refer to "LOWER-CASE and UPPER-CASE Function" in the "Syntax Samples".

Format

FUNCTION UPPER-CASE (argument-1)

Argument

1. The type of argument-1 must be alphabetic or alphanumeric.
2. The length of argument-1 must be one character or more.

Function Value

1. The function value is the character string of argument-1 with uppercase alphabetic characters converted to lowercase alphabetic characters. In other words, it is the same as argument-1, except that lowercase alphabetic characters in argument-1 have been converted to uppercase alphabetic characters.
2. The length of the function value is the same as of argument-1.

Function Type

The function type is alphanumeric.

F

6.6.49 UTF8-OF Function

This function converts the encoding forms of the characters included in argument-1 into UTF-8. The UTF8-OF function is available for [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

Format

FUNCTION UTF8-OF (argument-1)

Argument

1. Argument-1 must be a national data item.
2. Argument-1 must consist of one or more characters.
3. The encoding form of argument-1 must be UTF-16 or UCS-2.

Function Value

1. For the function value, the function converts the character indicated by argument-1 into the corresponding character string represented by UTF-8.
2. The size (bytes) of the function value follows the size of the converted character string.
3. The characters in argument-1 must be those that can be stored in a national data item. If another type of character is included, the results are undefined.
4. The encoding form of the function value is UTF-8.

Function Type

The function type is alphanumeric.

Rules specific toUTF-8 OF function

This function can be used only when the default code set is Unicode (ISO/IEC 10646_1).

6.6.50 VARIANCE Function

This function returns the approximate value of the variance of the elements of argument-1.

Format

FUNCTION VARIANCE ({argument-1}...)

Argument

The type of argument-1 must be numeric.

Function Value

1. The function value is the approximate value of the variance of argument-1. In other words, it is the square of the standard deviation of the list of argument-1 elements. For standard deviation, see the topic titled "STANDARD-DEVIATION Function above".
2. When all argument-1 elements have all the same value, the function value is zero.

Function Type

The function type is numeric.

6.6.51 WHEN-COMPLIED Function

This function returns the date and time at which the program was compiled.

For details about how to use the examples, refer to "WHEN-COMPILED Function" in the "Syntax Samples".

Format

FUNCTION WHEN-COMPILED

Function Value

1. The function value is 21 alphanumeric characters indicating the date and time when the program was compiled as well as the time difference from Greenwich Mean Time. The meaning of each of character in the function value is listed below.

Character Position	Value
1 to 4	4-digit number indicating the year
5 to 6	2-digit number indicating the month. The value ranges from 01 to 12.
7 to 8	2-digit number indicating the day. The value ranges from 01 to 31.
9 to 10	2-digit number indicating hours. The value ranges from 00 to 23.
11 to 12	2-digit number indicating minutes. The value ranges from 00 to 59.
13 to 14	2-digit number indicating seconds. The value ranges from 00 to 59.
15 to 16	2-digit number indicating 1/100 seconds. The value ranges from 00 to 99. (*1)
17	Sign indicating whether the local time (time indicated in character positions 1 to 16) is ahead or behind the Greenwich mean time. A minus (-) sign indicates that the local time is behind Greenwich mean time, and a plus (+) sign indicates that it is ahead. (*2)
18 to 19	2-digit number indicating the number of hours ahead or behind Greenwich mean time. The value ranges from 00 to 12 when character position 17 is "-"; and ranges from 00 to 13 when it is "+." (*2)
20 to 21	2-digit number indicating the number of minutes ahead or behind Greenwich mean time. The value ranges from 00 to 59. (*2)

*1 Character positions 15 and 16 are set to 00 if the system cannot read 1/100 seconds.

*2 Character positions 17 to 21 are set to 00000 if the system cannot read the time difference between the local time and Greenwich mean time.

2. The function value is the date and time at which the source program containing this function was compiled. The WHEN-COMPILED function in a nested program will return the date and time at which the highest nesting level program was compiled.

Function Type

The function type is alphanumeric.

Chapter 7 Source Text Manipulation

The source text manipulation function copies or replaces part of a source program from a COBOL library. It provides two statements: COPY and REPLACE.

The COPY statement copies library text into the program containing the COPY statement. When copying, it can also replace library text parts. The REPLACE statement replaces source program text parts.

The COPY and REPLACE statements can be written anywhere in the source program. A source program can consist of only COPY statements. At compilation time, COPY and REPLACE are processed before other statements are compiled. They have no function at execution time.

Text

A source program line or a set of lines and a COBOL library is called "text." Text in the source program is called "source program text", and text in the COBOL library is called "library text".

Text Word

Character strings in the COBOL library, source programs, and areas A and B in pseudo-text. The following character strings are called "text words":

- A separator, except for space, pseudo text delimiter, and separator quotation mark. The right and left parentheses of the separator are always handled as text words, regardless of their location.
- A literal is a text word. For
  - a nonnumeric literal,
  - F a national literal,
  - a Boolean literal,
  - and hexadecimal nonnumeric literal,
- the character strings between the starting and ending separators comprise one text word.
- Any character string between separators, which are, themselves, neither separators nor literals, is a text word. However, the comment line character string and the word COPY are not text words.

Pseudo-text

Any text word, comment line, and separator space between two pseudo-text delimiters in the source program and COBOL library are called "pseudo-text". A "pseudo-text delimiter" is two consecutive equal signs (==). Pseudo-text may not contain embedded pseudo-text delimiters.

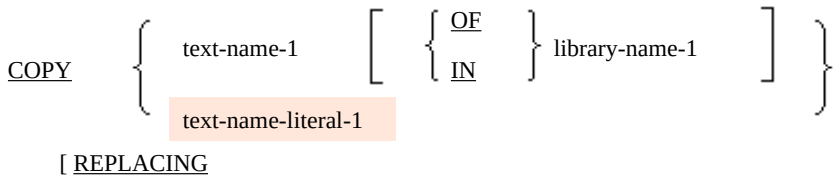
7.1 COPY Statement

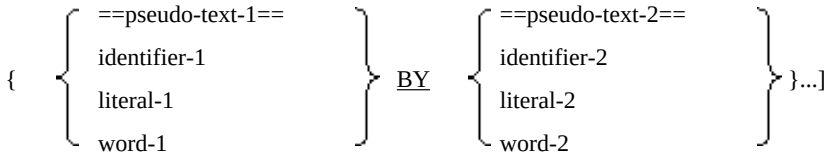
This statement copies library text to a source program.

For details about how to use the examples, refer to "COPY statement" in "Syntax Samples".

Format 1

Copies library text as it is, or partially modifies it when copying.

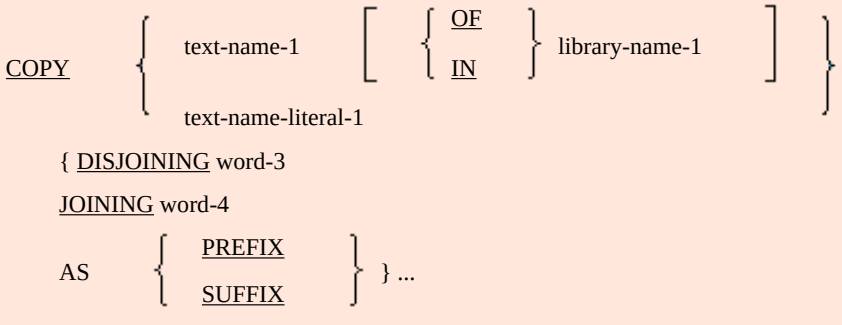




F

Format 2

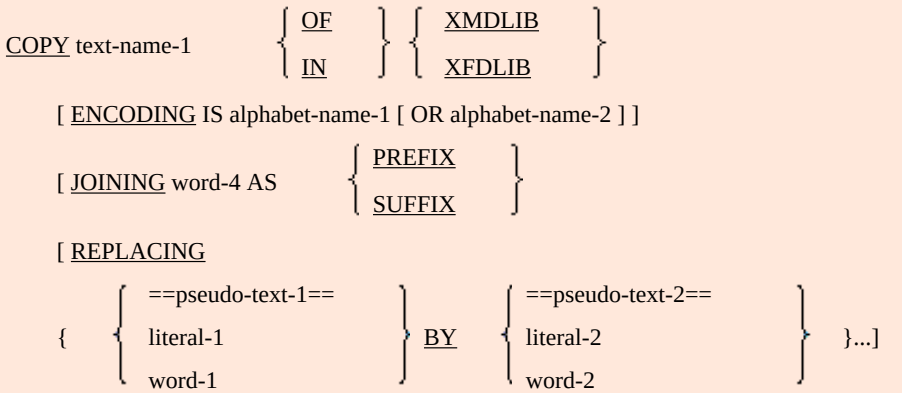
Replaces the prefix or suffix in library text to copy.



F

Format 3

For form and file descriptors, with option of adding a prefix or suffix to library text.



The ENCODING phrase in Format 3 is specific to [Win32], [Winx64] and [Linux64].

Syntax Rules

1. When two or more COBOL libraries are used in a compilation unit, text-name-1 should be qualified by the library name of the COBOL library containing the text. For information on COBOL text formatting, refer to the "COBOL User's Guide".
2. Text-name-1 must be unique in a COBOL library.
3. For text-name-1 and text-name-literal-1(*) description rules, see the section titled "Basic Overview of Language" in Chapter 1, "General Rules".
4. The COPY statement must begin after a space and end with a separator period.
5. Pseudo-text-1 and pseudo-text-2 must follow the reference format.
6. Pseudo-text-1 must contain more than one text word.
7. Pseudo-text-2 may consist only of the delimiters. In that case, the original text is deleted, i.e., replace by a null string.
8. Pseudo-text-1 and pseudo-text-2 can consist of more than one line.

9. Word-1, word-2, Word-3(*) and word-4(*) must be character strings following the COBOL word rules. However, the word COPY must not be specified.

F

   If national characters are used to specify word-3 and word-4, character-strings consisting of alphabetic-upper, alphabetic-lower, and numeric characters, JIS non-kanji characters can be used.

10. Embedded COPY statements are not permitted.
11. The length of pseudo-text-1, pseudo-text-2, and a text word in library text must be in the range of 1 to 324 characters.
12. Pseudo-text-1 must not consist of only a separator comma or semicolon.
13. The word COPY written in a comment item, comment line, or nonnumeric literal is handled as part of a character string comprising the comment item, the comment line, or the nonnumeric literal.

F

14. When user-defined words in library text use alphanumeric characters, word-4 must also be an alphanumeric character. When a user-defined word in library text is national language, word-4 must be national language.
15. For details on the maximum length of word-4, see Appendix B "System Quantitative Restrictions".
16. In Format 3, the JOINING phrase and the REPLACING phrase must not be specified concurrently.
17. In Format 3, alphabet-name-1 or alphabet-name-2 must be the alphabet-name associated with any of the following in the ALPHABET clause.
    - SJIS
    - UTF8
    - UTF16
    - UTF16LE
    - UTF16BE
    - UTF32
    - UTF32LE
    - UTF32BE
18. When alphabet-name-2 is specified in Format 3, the FOR phrase of the ALPHABET clause must not be the same as alphabet-name-1.

F

*: Extended functions or functions specific to NetCOBOL.

- text-name-literal-1
- Word-3 and word-4

General Rules

- Rules Common to All Formats

1. The COPY statement copies library text into a program at compilation time. Library text is specified in text-name-1 or text-name-literal-1(*).

   Each format supports different variations of copying the library text as specified in the detailed rules below. In overview the functions of the different formats is as follows:

   - When the REPLACING phrase is omitted in Format 1
     The COPY statement copies existing library text.
   - When the REPLACING phrase is specified in Format 1
     The COPY statement searches words in the library text for a character string matching the BY subject, then replaces any matching strings with the BY object before copying. Unmatched strings are copied unchanged.

F

   - When the PREFIX phrase is specified in Format 2
     The COPY statement searches library text for strings having the prefix in text words (strings from the left

end of a text word up to a hyphen) matching word-3 then replaces those matching strings with word-4 before copying.

When the SUFFIX phrase is specified in Format 2
The statement searches library text for strings having the suffix in text words (strings from the last hyphen up to the right end of the text word) matching word-3 then replaces matching strings with word-4 to before copying.

- Format 3 supports including data descriptions from PowerFORM form descriptors or FILE (a product only available in Japanese market) file descriptors. It also provides the ability to replace text as in Format 1 and to add a prefix or suffix to data-names. Because the compiler knows that the library text from XMDLIB and XFDLIB files consists only of data descriptions, the JOINING operations only operate on the data-names - the text immediately after the level numbers or the REDEFINES clause.

2. Source program compilation containing COPY statements is the same as source program compilation after processing all COPY statements. At compilation time, the entire COPY statement starting with COPY and ending with a period is replaced with library text or partially edited library text. The COPY statement has no function at execution time.
3. Library text must follow standard COBOL format.
4. COPY statements can be written in library text. However, no new COPY statement that could generate another COPY statement can be coded after the REPLACING phrase or the JOINING phrase(*) is processed.

F

* : Extended functions or functions specific to NetCOBOL.

- text-name-literal-1
- JOINING phrase

- Rules for Format 1

1. Text is replaced when the REPLACING phrase is specified. See the procedures below.
   a. Replacement comparison starts from the first text word other than separator commas or semicolons in library text. This text word is called "present text word". Spaces, separator commas, and semicolons placed before the text word are copied unchanged.
   b. The BY object is compared with library text beginning with the present text word, character by character. If two or more BY phrases are specified, all operands having the BY phrase are compared in the BY phrases specified order. Phrases are compared as follows:

      - When identifier-1, word-1, or literal-1 is specified, it is compared with the present text word.

      - When pseudo-text-1 is specified, the entire string in pseudo-text-1 is compared with library text beginning from the present text word. Each string of separator commas, semicolons, or spaces in pseudo-text-1 and library text is assumed to be a single blank.

   c. Step (b) is repeated until the comparison result matches or all operands on the left of BY phrase are processed.
   d. If no matching string is found in Step (b), the COPY statement copies the present text word unchanged. It then uses the next text word as "present text word" and returns to Step (b).
   e. When any matching string is found in Step (b), either of the following occurs:

      - When identifier-1, word-1, or literal-1 is specified, the COPY statement replaces the present text word with the BY object. Then, the next library text word is set as the present text word and the process returns to Step (b).

      - When pseudo-text-1 is specified, the COPY statement compares the strings matching pseudo-text-1 with the BY object. Then the last text word matching pseudo-text-1 is set as the present text word and the process returns to Step (b).

   f. Steps (b) to (e) are repeated until the final text word in library text is tested.
2. Comment line and blank lines in the library text and pseudo-text-1 are ignored during replacement comparison.

3. Library text, pseudo-text-1, and pseudo-text-2 can contain debugging lines. During comparison, "D" in the indicator area in library text and pseudo-text-1 is ignored.
4. Text words not replaced in the library text are copied and positioned in the same area as in library text. In other words, a text word starting in area A in library text is copied to start in area A and a text word starting in area B is copied to start in area B. However, if two or more text words exist in area A and one of them is replaced by a longer text word, it is copied to start in area B if it cannot start in area A.
5. Matching words in library text are replaced with a new text word (identifier-2, literal-2, word-2 or pseudo-text-2).
6. When text words in library text are replaced with identifier-2, literal-2, or word-2, the new text word (identifier-2, literal-2, or word-2) is copied to start from the same area in library text as the original text word.
7. When a text word in library text is replaced with the text in pseudo-text-2, each pseudo-text-2 text word is copied to start in the same area as pseudo-text-1. Spaces between text words in pseudo-text-2 are copied unchanged.
8. The following text words are copied to the debug line:
    a. When writing a COPY statement in a debug line, the text word in library text copied by the COPY statement.
    b. The text word in the library text debug line.
    c. The text word first replaced in debug line library text on the BY object.
    d. The text word in the pseudo-text-2 debug line.

    If the above text words are in a line containing the character "D" in the indicator area designating the debug line, the words are copied to the debug line even if a library text line crosses more than one line after text word replacement.
9. If literal-2, pseudo-text-2, or a literal contained in library text is continued in more than one line following the text word replaced by the COPY statement, the indicator area on the second and subsequent lines must contain "-" indicating line continuation. Debug lines, however, may not contain literal continuation.
10. If a new line is added during text replacement by the COPY statement, the new line indicator area reflects the same character as the line containing the original text word, except for case (8) and (9). If, however, the line indicator of the original line is a hyphen, a blank is set in the indicator area on the new line.
11. Comment lines and blank lines in library text are copied unchanged. However, a comment line or blank line in library text is not copied if that comment line or blank line appears within the sequence of text words that match pseudo-text-1.
12. Comment lines and blank lines in pseudo-text-2 are always copied when pseudo-text-2 is copied.

F

- Rules for Format 2

1. Text is replaced by the COPY statement in Format 2 as described in the following steps:
    a. Replacement comparison starts from the first text word other than any separator comma or semicolon in the library text. This text word is called the "present text word". Spaces, separator commas and separator semicolons before text words are copied unchanged.
    b. Word-3 is compared with the present text word. If there is more than one DISJOINING clause, all occurrences of word-3 are compared in the order written. Comparison is performed as follows:

    When word-3 is an alphanumeric string (not a National string), the present text word is inspected to see if it only contains alphanumeric characters and has word-3 as a prefix or suffix. When the PREFIX phrase is specified the comparison looks for word-3 plus a hyphen at the beginning of the present text word. When the SUFFIX phrase is specified the comparison looks for a hyphen plus word-3 at the end of the present text word.

    When word-3 is a National string, the present text word is inspected to see if it only contains National characters and has word-3 as a prefix or suffix. When the PREFIX phrase is specified the comparison looks for word-3 plus a JIS non-Kanji minus sign at the beginning of the present text word. When the SUFFIX phrase is specified the comparison looks for a JIS non-Kanji minus sign plus word-3 at the end of the present text word.

c. Step (b) is repeated until the comparison results in a match or all occurrences of word-3 are processed.

d. If no matching string is found in Step (b), the present text word is copied unchanged. The next text word is used as the "present text word" and returns to Step (b).

e. When any matching string is found in Step (b) and copied, the present text word matching word-3 is replaced with word-4. Then, the next library text word is set as the present text word and processing returns to Step (b).

f. Steps (b) to (e) are repeated until the last text word in the library set is compared.

2. Matching text words in library text are replaced with a new text word (prefix or suffix replaced with word-4) and copied according to the reference format.

3. When text words in library text are replaced, new text words (prefix or suffix replaced with word-4) are copied to start from the same area started in the library text.

F

- Rules for Format 3

1. Text-name-1 is treated as a screen form descriptor name when XMDLIB is specified. It is treated as a file descriptor name when XFDLIB is specified.
XFDLIB is a capability native to [UXP/DS], [Win16], [Solaris], and [.NET].

2. For the JOINING clause, all data-names immediately after the level number or in the library text REDEFINES clause are replaced according to the rules below. The "old data-name" represents all data-names written immediately after a level number or in the REDEFINES clause.

   a. When word-4 is an alphanumeric string (not a National string), the old data-name must also be an alphanumeric string. The PREFIX phrase replaces and copies the old data-name with word-4 concatenated with a hyphen plus the old data-name. The SUFFIX phrase replaces and copies the old data-name with the old data-name concatenated with a hyphen plus word-4.

   b. When word-4 is a National string, the old data-name must be a National string. The PREFIX phrase replaces and copies the old data-name with word-4 concatenated with a JIS non-Kanji minus sign plus the old data-name. The SUFFIX phrase replaces and copies the old data-name with the old data-name concatenated with a JIS non-Kanji minus sign plus word-4.

   c. Other text words are copied unchanged.

3. When text words in library text are replaced, new text words (prefix or suffix replaced with word-4) are copied and start from the same area as started in library text t.

4. When text words in library text are replaced, new text words (prefix or suffix replaced with word-4) are copied and start from the same area as started in library text.

5. The REPLACING clause acts as described for Format 1.

6. When ALPHANUMERIC is specified in the alphabet name explicitly or implicitly, then the encoding form is applied in the alphanumeric data item or alphanumeric edited data item.

7. When NATIONAL is specified in the alphabet-name, then the encoding form is applied to the National data item or National edited data item.

7.2 REPLACE Statement

This statement replaces source program text.

Format 1

Specifies the character string to be replaced and declares start of replacement

REPLACE {== pseudo-text-1 == BY == pseudo-text-2 ==} ...

Format 2

Declares end of text replacement

REPLACE OFF

Syntax Rules

1. The REPLACE statement can be written in any position where character strings can be written in the source program. It must follow a separator period, except when it is the first statement in a separately compiled program.
2. The REPLACE statement must end with a separator period.
3. Pseudo-text-1 and pseudo-text-2 must follow standard COBOL formats.
4. Pseudo-text-1 must contain one or more text words.
5. Pseudo-text-2 can contain one or more text words. Text words may be omitted, resulting in the replacement of pseudo-text-1 by a null string.
6. Character strings in pseudo-text-1 and pseudo-text-2 may contain continuation lines using standard COBOL format.
7. The length of each text word in pseudo-text-1 and pseudo-text-2 must range from 1 to 324 characters.
8. Pseudo-text-1 must not consist of only separator commas or semicolons.
9. If the word REPLACE is specified in a comment item, comment line, or nonnumeric literal, the word REPLACE is handled as part of the character string of the comment item, comment line, or nonnumeric literal.

General Rules

1. The REPLACE statement replaces source program text at compilation time. The REPLACE statement in Format 1 specifies replacing any character string matching pseudo-text-1 in the source program with pseudo-text-2 and declares the start of text replacement. The REPLACE statement in Format 2 declares the end of text replacement. The replacement rules for the REPLACE statement in Format 1 are valid until the next REPLACE statement appears or up to the end of the separately compiled program.
2. The REPLACE statement is processed after the COPY statement at compilation time. It has no function at run time.
3. No REPLACE statements can be specified to generate new REPLACE statements during execution of the REPLACE statement. In other words, nested REPLACE is not allowed.
4. Text is replaced using the procedures below.
    a. The REPLACE statement starts replacement comparison from the first text word in the source program. This text word is called "present text word".
    b. The entire string in pseudo-text-1 is compared with the source program text starting from the present text word comparing character by character. If pseudo-text-1 is written more than once, all pseudo-text-1 elements are compared in the order they appear. At this time, each string of separator commas, semicolons, and separator spaces in pseudo-text-1 and the source program text is assumed to be one space.
    c. Step (b) is repeated until a matching string is found or all the pseudo-text-1 elements are processed.
    d. If no matching string is found in Step (b), REPLACE sets the next text word as the present text word then returns to Step (b).
    e. If any matching string is found in Step (b), REPLACE replaces strings matching pseudo-text-1 with pseudo-text-2 according to the reference format. Then, it sets the text word following the last string matching pseudo-text-1 as present text word and returns to Step (b).
    f. The REPLACE statement repeats Steps (b) to (e) until the last text word in library text.
5. The REPLACE statement ignores any comment or blank line in the source program and pseudo-text-1 during replacement comparison.
6. A debug line can be written in pseudo-text-1 and pseudo-text-2. The statement ignores "D" in the indicator area in pseudo-text-1 and pseudo-text-2 during comparison.
7. REPLACE replaces strings matching pseudo-text-1 in the source program text with pseudo-text-2 and inserts each pseudo-text-2 text word according to the reference format. It inserts spaces between pseudo-text-1 text words.

8. If literals in pseudo-text-2 are inserted across more than one line during text replacement by the REPLACE statement, "-" is set in the indicator area on the next line to indicate line continuation. However, a literal inserted on a debug line must not continue on the next line.
9. If new lines are added by the REPLACE statement text replacement, the same character as the line indicator area containing the old text is set in the new line indicator area. However, if the line indicator area contains a hyphen, a blank is set in the new line indicator area.
10. Any comment or blank line in pseudo-text-2 is inserted unchanged whenever pseudo-text-2 is inserted.
11. The REPLACE statement must not change the PROGRAM-ID paragraph and END PROGRAM header.

F

F

Chapter 8 Database (SQL)

The database function (SQL) can access various types of databases and manipulate data by using embedded SQL statements as defined in the COBOL compiler.

The database (SQL) function is specific to [Win16], [Win32], [Winx64], [Linux], and [.NET].

Database

A relational database is a set of two-dimensional tables. A table contains a set of rows and columns.

Data in a database can be sorted dynamically by values indicated by the user. The user can access data in a table, regardless of the physical location and order of the data. Data is processed by using a table as a logical user interface.

Table

Actual tables and view tables are generically referred to as tables. Data is stored in actual tables. View tables are virtual tables used for data manipulation. Virtual tables do not exist as physical data.

Embedded SQL

Embedded SQL is designed to manipulate databases within application programs. Embedded SQL contains the following:

- Embedded SQL declarative section

  This section starts with an embedded SQL BEGIN DECLARE statement and ends with embedded SQL END DECLARE statement. Host variables are defined in the embedded SQL declarative section.

- Embedded SQL statement

  This statement manipulates databases and describes exception handling.

  The embedded SQL statements defined in this chapter are the COBOL compiler-defined embedded SQL statements. Embedded SQL statement operation details and its description depend on the particular database specifications.

Host Variable

Host variables are data items used for sending and receiving data between databases and application programs.

Host variables can be classified into two types. One is an elementary item that corresponds to a single column in a database (host variable with a single column specified). Another is a group item that corresponds to multiple columns in the database (host variable with multiple columns specified, host variable with multiple rows specified, or host variable with a table specified).

8.1 Format of Embedded SQL

8.1.1 Overall Rules

The embedded SQL BEGIN DECLARE, embedded SQL END DECLARE, and embedded SQL statement must be enclosed by the SQL prefix ("EXEC SQL") and the SQL terminator ("END-EXEC").

The embedded SQL BEGIN DECLARE, embedded SQL END DECLARE, and embedded SQL statement must all be described in Area B.

8.1.2 Continuation of a Line

The continuation rule (continuation of line) of an embedded SQL BEGIN DECLARE, embedded SQL END DECLARE, and embedded SQL statement conforms to the COBOL format rule.

8.1.3 COBOL Comment Line and In-line Comment

COBOL comment lines, debugging lines, and blank lines can be included in embedded SQL; however, in-line comments cannot.

8.2 Data Division

In the DATA DIVISION, the embedded SQL declare section describes where host variables are defined.

8.2.1 Embedded SQL Declare Section

Host variables are declared in the embedded SQL declare section.

Format

EXEC SQL

BEGIN DECLARE SECTION END-EXEC.

[{ host variable definition } ...]

EXEC SQL

END DECLARE SECTION END-EXEC.

General Rules

1. An embedded SQL declare section can be included in the following:
   - WORKING-STORAGE SECTION of the DATA DIVISION
   - LINKAGE SECTION of the DATA DIVISION
   - LOCAL-STORAGE SECTION of the DATA DIVISION ([Winx64] and [.NET] only)
2. An embedded SQL BEGIN DECLARE (EXEC SQL BEGIN DECLARE SECTION END-EXEC) and an embedded SQL END DECLARE (EXEC SQL END DECLARE SECTION END-EXEC) cannot be omitted.
3. An embedded SQL BEGIN DECLARE must be paired with an embedded SQL END DECLARE. They may not be nested.
4. A data item must not be redefined across an embedded SQL BEGIN DECLARE or SQL END DECLARE.
5. An embedded SQL declare section may not be included in an embedded DCSQL declare section. An embedded DCSQL declare section may not be included in an embedded SQL declarative section. This rule is native to [Win16].

8.2.2 Host Variable Definitions

Host variable definitions specify the host variable characteristics.

Format

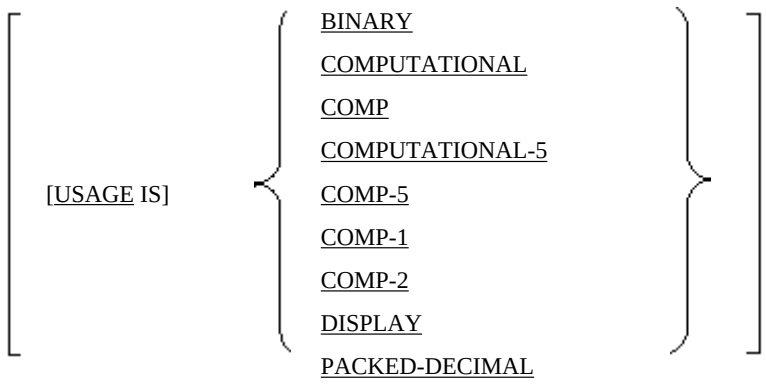
level-number variable-name

[REDEFINES clause]

[IS EXTERNAL]

[IS GLOBAL]

[{ PICTURE / PIC } IS character-string]



[[SIGN IS]

{ LEADING SEPARATE CHARACTER / TRAILING }]

[OCCURS integer-1 [TIMES]].

The REDEFINES clause is specific to [Win32], [Winx64] and [.NET].

Syntax Rules

1. A host variable must be an item with a level number from 01 to 48 or 77.
2. A host variable corresponding to a variable-length string data in a database must be a group item subordinating an elementary item with level number 49.
3. When the OCCURS clause is specified in an elementary item, the elementary item must not have level number 49. When the OCCURS clause is specified in a group item, the group item must not correspond to a variable-length string data in a database.
4. When the REDEFINES clause is specified, its level number must not be 49.
5. A host variable must not be a variable address item.
6. The syntax rules of each clause are the same as each clause in the COBOL data description entry.

General Rules

1. The general rules of each clause are the same as each clause of the COBOL data description entry.

   However, when host variable names subordinate to a group item and the group is a global name, it must be a unique name without qualification within its scope.

2. Host variable must be one of the following data types:

   a. In [Win16], [Win32], [Linux] and [.NET], Signed binary integer data item of 4 or 9 digits
   In [Winx64] and in 18-digit compatible mode, Signed binary integer data item of 4 or 9 digits
   In [Winx64] and in 31-digit extension mode, Signed binary integer data item of 4, 9 or 18 digits

   b. In [Win16], [Win32], [Linux] and [.NET], Signed packed decimal data item of 18 or less digits
   In [Winx64] and in 18-digit compatible mode, Signed packed decimal data item of 18 or less digits
   In [Winx64] and in 31-digit extension mode, Signed packed decimal data item of 31 or less digits

   c. In [Win16], [Win32], [Linux] and [.NET], Signed zoned decimal data item of 18 or less digits
   In [Winx64] and in 18-digit compatible mode, Signed zoned decimal data item of 18 or less digits
   In [Winx64] and in 31-digit extension mode, Signed zoned decimal data item of 31 or less digits

   d. Alphanumeric data item

      Refer to "NetCOBOL User's Guide" for the details of the item length.

   e. National data item

      Refer to "NetCOBOL User's Guide" for the details of the item length.

f. Single-precision internal floating-point data item

g. Double-precision internal floating-point data item

h. Variable-length string type group items of the following data types and structures (refer to the "NetCOBOL User's Guide" for the m or n value):

```
       01  data-name-1.
                49  data-name-2 PIC S9(m) BINARY.
                49  data-name-3 PIC X(n).
```

or

```
       01  data-name-1.
                49  data-name-2 PIC S9(m) COMP-5.
                49  data-name-3 PIC X(n).
```

or

```
       01  data-name-1.
                49  data-name-2 PIC S9(m) BINARY.
                49  data-name-3 PIC N(n).
```

or

```
       01  data-name-1.
                49  data-name-2 PIC S9(m) COMP-5.
                49  data-name-3 PIC N(n).
```

3. Whether national data items can be used depends on the database and its related products.

4. A host variable with multiple columns specified corresponds to multiple columns in a database.

```
level-number-1 data-name-1.  <-    Host variable with multiple columns specified
    host-variable-with-single-column-specified-1
    host-variable-with-single-column-specified-2
                :
    host-variable-with-single-column-specified-x
```

5. A host variable with multiple rows specified is an elementary item having the OCCURS clause. It corresponds to multiple rows in a database.

```
level-number-1 data-name-1.
   host-variable-with-single-column-specified OCCURS integer-1  <- Host variable with
multiple rows specified
```

6. A host variable with a table specified is a group item that can contain only host variables of multiple rows specified. It corresponds to a database table consisting of columns and rows.

```
level-number-1 data-name-1.     <-   Host variable with a table specified
    host-variable-with-multiple-rows-specified-1
    host-variable-with-multiple-rows-specified-2
                    :
    host-variable-with-multiple-rows-specified-x
```

7. When a host variable with multiple rows specified or a host variable with a table specified is used with the target specification, specify the number of rows of data to be fetched in integer-1. When a host variable with multiple rows specified or a host variable with a table specified is used with the value specification, specify the number of times the SQL statement is to be executed in integer-1.

8. The value in integer-1 of an item subordinate to a host variable with a table specified is regarded as the number of occurrence of this host variable with a table specified.

9. The values of integer-1 must be the same among the items subordinate to a host variable with a table specified.

8.2.3 Referencing Host Variables

Host variables can be referenced either in embedded SQL statements or in general COBOL statements. If a host variable is referenced in an embedded SQL statement, a colon (:) must be added to the front of the host variable name. For example, if a host variable A is referenced, it must be declared as :A in the embedded SQL statement.

If a host variable is referenced in a general COBOL statement, do not add the colon.

SQLSTATE, SQLCODE, SQLMSG, or SQLERRD must not be referenced as a host variable in embedded SQL.

8.2.4 SQLSTATE/SQLCODE

If an exception event occurs during the execution of an SQL statement, a code indicating the nature of the event is posted to the application program. SQLSTATE/SQLCODE is the area where the code is stored. The user can take appropriate action for the exception event by referencing SQLSTATE/SQLCODE.

SQLSTATE

- In [Win16] and [Linux], SQLSTATE must always be defined as a five-digit alphanumeric data item with level number 01 or 77 in the embedded SQL declarative section.
- SQLSTATE must always be defined as a 5-digit alphanumeric item that must be declared as a level 01 or level 77 item. In [Win32], [Winx64] and [.NET], SQLSTATE can be declared as a data item subordinate to a level 01 data item named SQLCA. In this case, SQLSTATE must be an 02 through 49 level item. SQLSTATE must not be declared as both a level 01 or level 77 item and as a data item subordinate to SQLCA.
- The REDEFINES clause cannot be specified for SQLSTATE.

SQLCODE

- In [Win16] and [Linux], SQLCODE must always be defined as a nine-digit signed binary integer data item with level number 01 or 77 in the embedded SQL declarative section.
- SQLCODE must always be defined as a 9-digit alphanumeric item that must be declared as a level 01 or level 77 item. In [Win32], [Winx64] and [.NET], SQLCODE can be declared as a data item subordinate to a level 01 data item named SQLCA. In this case, SQLCODE must be an 02 through 49 level item. SQLCODE must not be declared as both a level 01 or level 77 item and as a data item subordinate to SQLCA.
- The REDEFINES clause cannot be specified for SQLCODE.

8.2.5 SQLMSG

If an exception event occurs during the execution of an SQL statement, a message indicating the nature of the event is posted to the application program. SQLMSG is the area where this message is stored. The user can print or display the SQLMSG contents by using an output statement.

- In [Win16] and [Linux], SQLMSG must always be defined as an alphanumeric data item that must be declared as a level number 01 or level 77 item and must be defined in the embedded SQL declarative section.
- In [Win32], [Winx64] and [.NET], SQLMSG must always be defined as an alphanumeric data item that must be declared as a level number 01 or level 77 item.
- The REDEFINES clause cannot be specified for SQLMSG.

8.2.6 SQLERRD

SQLERRD is used to store the following items:

- The return value of the stored procedure

  In [Win32], [Winx64] and [.NET], refer to SQLERRD(1), which is the first element of the array, to view the return value of the stored procedure.

- Processed rows

  When multiple rows are processed using a host variable with multiple rows specified, the number of processed rows is reported to the application program. The user can determine the number of processed rows by referencing SQLERRD (3), the third element of the array.

Format 1

```
01 SQLINF0A.
    02 SQLERRD PIC S9 (9)    { COMP   }
                             { COMP-5 }
                             { BINARY }
                OCCURS 6 [TIMES].
```

Format 2

```
01 SQLCA.
    Level-number SQLERRD PIC S9 (9)    { COMP   }
                                       { COMP-5 }
                                       { BINARY }
                OCCURS 6 [TIMES].
```

Format 2 is for functions that are specific to [Win32], [Winx64] and [.NET].

Syntax Rules

Rules Common to Format 1 and Format 2

In [Win16] and [Linux], SQLERRD must be defined in the embedded SQL declarative section.

SQLERRD cannot be defined as subordinate to both SQLINFOA and SQLCA.

Rules for Format 2

In [Win32], [Winx64] and [.NET], SQLERRD can be declared as a data item subordinate to a level 01 data item named SQLCA. In this case, the level number for SQLERRD must be an 02 through49 level item.

General Rules

The return value of the stored procedure is posted to SQLERRD (1) and the number of processed rows is posted to SQLERRD (3). The other areas are reserved for the system and must not be used. SQLERRD (1) is specific to [Win32], [Winx64] and [.NET].

8.3 Procedure Division

In the PROCEDURE DIVISION, embedded SQL statements are described.

The following symbols are used in this section:

- <> : Name of an element forming a statement
- ::= : Operator forming an element. The element to be defined appears to the left of the operator and the formula defining the element appears to the right of the operator.
- [] : Optional elements
- {} : The user selects one of the elements described in the {}. The selectable elements are either separated by a vertical bar ("|") or written vertically.
- |: The element written after the vertical bar can be used instead of the element before the vertical bar. The vertical bar is given in braces.

- ... : The preceding element is repeated.

8.3.1 Character

The following characters may be used in SQL statements:

- Alphabetic character
- Digit
- Special character
- National character

Alphabetic Character

See the section titled "Characters and Character Sets" of Chapter 1, "General Rules" for information on alphabetic characters.

Digit

See the section titled "Characters and Character Sets" of Chapter 1, "General Rules" for information on digits.

Special Character

The following 18 special characters are available:

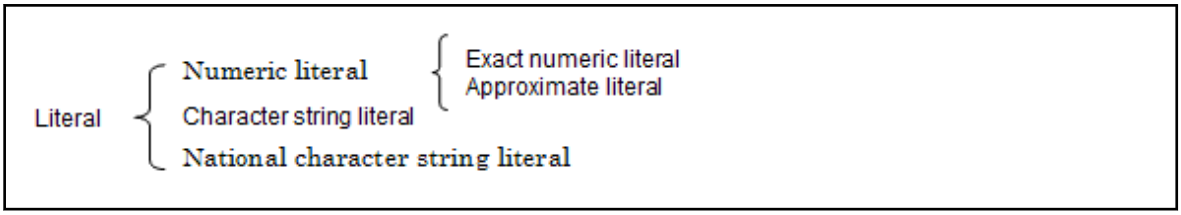
```
., <, (, +, *, ), ;, -, /, ,, %, _, >, ?, :, =, ", '
```

National Character

Whether national characters can be used depends on the database and related products.

8.3.2 Literals

Values, excluding null value, are specified as literals in the embedded SQL statement. The following literal types are available:



8.3.2.1 Numeric Literal

A numeric literal has a numeric value.

Format

<numeric literal> ::= <exact numeric literal> | <approximate literal>

<exact numeric literal> ::=

[+|-]{ <unsigned integer>[.<unsigned integer>] | <unsigned integer>.|.<unsigned integer>}

<approximate literal> ::= <mantissa>E<exponent>

<mantissa> ::= <exact numeric literal>

<exponent> ::= [+|-]<unsigned integer>

<unsigned integer> ::= <digit> ...

Referenced section

Element	Section
<digit>	"Character"

Syntax Rules

1. The exact numeric literal is a character string containing the digit 0 to 9 and a decimal point.
2. A positive or negative sign can be specified for the exact numeric literal.
3. The exact numeric literal data type is exact numeric. The precision of exact numeric literal is the number of digits in the literal. The exact numeric literal scale is the number of digits at the right of the decimal point.
4. The precision and scale of exact numeric literals conform to the rules of the exact numeric host variable.
5. The approximate literal is a character string containing the digits 0 to 9, a decimal point, and the character E.
6. A positive or negative sign can be specified for approximate literal.
7. The approximate literal data type is approximate. The precision of approximate literal is of its mantissa. The mantissa is specified in the same format as exact numeric literal.
8. The precision and scale of approximate literal conform to the approximate host variable.

General Rules

1. An exact numeric literal represents fixed-point numeric data.
2. The interpreted numeric value of an exact numeric literal is derived from the normal mathematical interpretation of signed positional decimal notation.
3. An approximate literal represents floating-point numeric data.
4. The interpreted numeric value of an approximate literal is approximately the product of the exact numeric value, represented by the mantissa, with the number obtained by raising the number 10 to the power represented by the exponent.

8.3.2.2 Character String Literal

A character string literal has characters as its value.

Format

'<character>...'

Referenced section

Element	Section
<character >	"Character"

Syntax Rules

1. A character string literal is a character string starting and ending with a quotation mark. This literal specifies character data.
2. The character string literal data type is character string.
3. The character string literal length conforms to the rule for the host variable length whose data type is character string.

General Rules

The character string literal value is the characters series the literal contains between the quotation marks.

8.3.2.3 National Character String Literal

A national character string literal is a literal having national characters as its value. Whether the national character string literal can be used depends on the database and their related products.

Format

N'<national character>...'

Referenced section

Element	Section
<national character>	"Character"

Syntax Rules

1. A national character string literal is a national character string that starts with a list of N and a quotation mark and ends with a quotation mark. This literal specifies each national character.
2. The national character string literal data type is national character string. The national character string literal length is the number of national characters included in the literal.
3. The national character string literal length conforms to the rule of the host variable length whose data type is a national character string type.

General Rules

The national character string literal value is the series of national characters the literal contains.

8.3.3 Token

The token specifies the minimum unit for constructing an SQL statement.

Format

<token> ::= <non-delimiting token>|<delimiting token>

<non-delimiting token> ::= <identifier>|<key word>|<numeric literal>

<identifier> ::= <alphabetic character>[{[<underscore>] {<alphabetic character>|<digit>}}...]

| <national character identifier>

<national character identifier> ::= <national character string literal>

<underscore> ::= _

<key word> ::= Refer to "NetCOBOL User's Guide."

<delimiting token> ::= <character string literal>|<national character string literal>

|,|(|)|<|>|.|:|=|*|+|-|/|<>|>=|<=

<separator> ::= {<comment>|<space>|<new-line>}...

<comment> ::= <comment introducer> [{<character>|<national character>}...]

<comment introducer> ::= --

Referenced section

Element	Section
<alphabetic character>	"Character"
<digit>	"Character"

Element	Section
<character>	"Character"
<national character>	"Character"
<numeric literal>	"Numeric literal"
<character string literal>	"Character string literal"
<national character string literal>	"National character string literal"

Syntax Rules

The comment introducer is a list of two or more consecutive hyphens not separated by any spaces or new lines that is not included as a literal.

8.3.4 Names

The following six types of names are available in embedded SQL names:

- Table name
- Cursor name
- Column name
- Correlation name
- SQL statement identifier
- Stored procedure name

8.3.4.1 Table Name

The table name is the name of an actual table. The table name is used to specify the table used for data manipulation. The table name can be qualified by separating the qualified table from the qualifying table by inserting a period (.).

8.3.4.2 Cursor Name

A cursor name is assigned to a cursor. The cursor is the row indicator specifying a row in a table. The name is defined in a cursor declarative.

A cursor name must be an alphanumeric or a national character string containing 18 characters or less.

8.3.4.3 Column Name

The column name is the name of a column in a table. The column name can be qualified by the table or correlation name. The column name is qualified by separating the qualified column from the qualifying name by inserting a period ".".

8.3.4.4 Correlation Name

A correlation name is an alias of a table.

8.3.4.5 SQL Statement Identifier

The SQL statement identifier is a statement identifier used in a dynamic SQL statement. The SQL statement identifier is an alphanumeric or a national character string containing 18 characters or less.

8.3.4.6 Stored Procedure Names

Stored procedure names are defined in a stored procedure definition. Stored procedure names can be qualified. To qualify a stored procedure name, use a period "." to separate the stored procedure name to be qualified and a qualifying name. A stored procedure name must be a list of alphanumeric characters of the appropriate language that does not exceed 90 characters.

8.3.5 Value Specification and Target Specification

A value specification specifies a host variable, an indicator variable, or a literal.

A target specification specifies the host variable and the indicator variable assigned to the value.

Format

<value specification> ::= <variable specification> | <literal>

<variable specification> ::= <host variable name> [<indicator variable>]

<dynamic parameter specification> ::= ?

<indicator variable> ::= [INDICATOR] <host variable name>

<host variable name> ::= [{<host variable>.}...] <host variable>

<host variable> ::= See each rule.

<target specification> ::= <variable specification>

Referenced section

Element	Section
<literal>	"Literals"

Syntax Rules

1. The host variable name is the name given to a host variable. The host variable can be declared where an identifier or a data-name can be declared in an embedded SQL statement and in COBOL ordinary format.
2. The host variable name must be an alphanumeric or a national character string containing 30 characters or less. Its structure conforms to the COBOL user-defined words rule.
3. The host variable must be defined in an embedded SQL declare section.
4. A host variable cannot be reference modified or subscripted in an SQL statement.
5. When a host variable with multiple columns or a host variable with a table specified is specified, it is assumed that all subordinate items are specified as host variables in the order of definition.
6. A host variable can be qualified to make it unique. To qualify a host variable in the SQL statement, write the qualifier before the qualified name delimited by a period.
7. A qualifier must be the name of the group item that directly subordinates the qualified item.
8. If the name is unique, the higher level names within the hierarchy need not be specified as a qualifier.
9. A host variable specified by an indicator variable becomes an indicator variable.
10. The indicator variable must be a 4 digit signed binary integer data item.
11. The indicator variable can be specified by pairing it with a host variable to be stored in a table column or with a host variable to read a value from the column.
12. The dynamic parameter specification is used to specify a variable as a value specification in a statement being prepared. The dynamic parameter specification specifies a question mark with a specified value. A dynamic parameter specification in a prepared statement is equivalent to a variable specification in an embedded SQL statement.

General Rules

1. A host variable with multiple columns specified or a host variable with a table specified can be used in the following SQL statements:
   - Target specification of a SELECT statement
   - INSERT statement

- FETCH statement
- USING and INTO clauses of an EXECUTE statement

2. A host variable with multiple rows specified can be used in the following SQL statements:
   - Target specification of a SELECT statement
   - INSERT statement
   - FETCH statement
   - SET and WHERE clauses of an UPDATE statement (searched)
   - DELETE statement (searched)
   - USING and INTO clauses of an EXECUTE statement
3. Where a host variable with multiple rows can be specified, the host variable must not be used together with a host variable with a single row or a host variable with multiple columns.
4. Where a host variable with a table can be specified, it must not be used together with a host variable with a single row or a host variable with multiple columns.
5. The occurrence number specified in the subordinate items of a host variable with multiple rows specified and host variable with a table specified used in one SQL statement must all be the same. If they are not the same, the smaller number is used.
6. When a host variable value related by an indicator variable is stored in a table column, whether null value is stored can be controlled by setting the following values for the indicator variable:
   - 0 or a positive value: The host variable value related to the indicator variable is stored in the table column.
   - Negative value: A null value is stored in the table column. In this case, the contents of the host variable related to the indicator variable are ignored.
7. When a value is read from a column into a host variable related to an indicator variable, the following values are stored in the indicator variable by the database system after the SQL statement is executed:
   - 0 or a positive value: The input value is not a null value and is stored in the host variable related to the indicator variable.
   - Negative value: The input value is a null value. In this case, the contents of the host variable related to the indicator variable have no meaning.
8. The indicator variable corresponding to a host variable with multiple columns must be a host variable with multiple columns that has as at least as many subordinate items as the host variable has, or must be an elementary item with OCCURS clause that specifies a number of occurrence which is at least as many as the number of subordinate items of the host variable.
9. The indicator variable corresponding to a host variable with multiple rows must be a host variable with multiple rows that specifies at least as many occurrences as the host variable.
10. The indicator variable corresponding to a host variable with a table specified must be a host variable with a table specified that has at least as many subordinate items and occurrences as the host variable.
11. The subordinate items of an indicator variable are associated with the subordinate items of a host variable in order of definition of the latter subordinate items. An array is associated in order from the first array element.
12. When the number of subordinate items or occurrences of an indicator variable is greater than that of a host variable, the remaining indicator variables are ignored.

8.3.6 Column Specification

A column specification references the column specified by the column name. See the topic titled "Column name" of the "Procedure Division" section earlier in this chapter for information on column names.

8.3.7 Set Function Specification

The set function specification describes a value determined by applying a function.

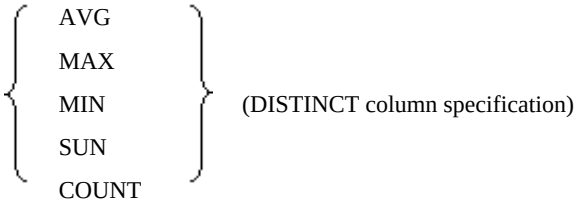
Format 1

Returns the total number of rows in a table

COUNT(*)

Format 2

DISTINCT set function



Format 3

ALL set function

{AVG|MAX|MIN|SUM} ([ALL] value expression)

Referenced section

Element	Section
<column specification>	Column Specification
<value expression>	Value Expression

General Rules

1. The format 1 set function returns the total number of rows, including columns having null value.
2. The DISTINCT set function returns the result obtained by excluding columns having a null value and counting columns with a unique value.
3. The ALL set function returns the result obtained by excluding columns having a null value.
4. The functions available in format 2 and 3 are:
   - AVG: Obtains the average value.
   - MAX: Obtains the maximum value.
   - MIN: Obtains the minimum value.
   - SUM: Obtains the sum.
   - COUNT: Obtains the cardinal number of a table.

8.3.8 Value Expression

A value expression specifies a value.

Format

<value expression> ::= <term>|<value expression>+<term> | <value expression>-<term>

<term> ::= <factor>|<term>*<factor>|<term>/<factor>

<factor> ::= [+|-]<primary>

<primary> :: = <column specification>

| <dynamic parameter specification>|<literal> | <set function specification>|(<value expression>)

Referenced section

Element	Section
<column specification>	"Column specification"
<dynamic parameter specification>	"Value specification and Target specification"
<literal>	"Literals"
<set function specification>	"Set function specification"

Syntax Rules

Do not use successive unary arithmetic operators.

General Rules

1. An operator indicating exact or approximate numeric can specified for a literal.
2. The unary plus (+) does not affect the element immediately following. The unary minus (-) indicates the element immediately following is to be multiplied by -1.
3. Binary arithmetic operators +, -, *, and / indicate addition subtraction, multiplication, and division respectively.
4. Parenthesized expressions, unary arithmetic operators, multiplication and division, and addition and subtraction are evaluated in this order. Operators at the same level are evaluated from left to right.
5. A divisor may not be 0.

8.3.9 Predicate

A predicate specifies conditions for creating a truth table including true, false, and unknown. A predicate has the following types:

- Comparison predicate
- BETWEEN predicate
- IN predicate
- LIKE predicate
- NULL predicate
- Quantified predicate
- EXISTS predicate

8.3.9.1 Comparison Predicate

Specify comparison of two row values.

Format

<comparison predicate> ::= <value expression><comp op>{<value expression> | <subquery>}

<comp op> ::= {=|<|<=|>|>=|<>}

Referenced section}

Element	Section
<value expression>	"Value expression"
<subquery>	"Subquery"

General Rules

The first value expression data type and the second value expression or subquery must be respectively comparable. The degree of the subquery result must be 1.

8.3.9.2 BETWEEN Predicate

Specify a range comparison.

Format

<BETWEEN predicate> ::=

<value expression> [NOT] BETWEEN <value expression> AND <value expression>

Referenced section

Element	Section
<value expression>	"Value expression"

General Rules

1. The three value expression data types must be respectively comparable.
2. Variable specifications and dynamic parameter specifications must not be included in the value expressions.

8.3.9.3 IN Predicate

Specify a quantified comparison.

Format

<IN predicate> ::= <value expression> [NOT] IN {(<subquery>) | (<in value list>)}

<in value list> ::= <in value>{,<in value>}...

<in value> ::= {<literal>|<variable specification>|USER|<dynamic parameter specification>}

Referenced section

Element	Section
<value expression>	"Value expression"
<subquery>	"Subquery"
<literal>	"Literals"
<variable specification>	"Value specification and Target specification"
<dynamic parameter specification>	"Value specification and Target specification"

General Rules

The first value expression data type and the value specification in the subquery or in the value list must be respectively comparable.

8.3.9.4 LIKE Predicate

Specify a pattern-match comparison in the character type data.

Format

<LIKE predicate> ::= <mach value> [NOT] LIKE <pattern>

<match value> ::= <column specification>

<pattern> ::= <character string literal>|<variable specification>|USER|

<dynamic parameter specification>

Referenced section

Element	Section
<column specification>	"Column specification"
<character string literal>	"Literals"
<variable specification>	"Value specification and Target specification"
<dynamic parameter specification>	"Value specification and Target specification"

General Rules

1. The column data type specified by column specification must be character or national character string type.
2. If the column data type specified by the column specification is character string type, a character string must be specified as the pattern. If the column data type specified by the column specification is national character string type, a national character string must be specified as the pattern.
3. The pattern can consist of a character string and characters % (percent) and _ (underscore). It can also consist of a national character string and the national characters "%" and "_".
4. The percent character corresponds to 0 or more arbitrary characters.
5. The underscore character corresponds to one arbitrary character.

8.3.9.5 NULL Predicate

Specify a test for a null value.

Format

<NULL predicate > ::= <column specification> IS [NOT] NULL

Referenced section

Element	Section
<column specification>	"Column specification"

8.3.9.6 Quantified Predicate

Specify a quantified comparison.

Format

<quantified predicate> ::= <value expression> <comp op> {ALL|ANY} <subquery>

Referenced section

Element	Section
<value expression>	"Value expression"
<comp op>	"Comparison predicate"
<subquery>	"Subquery"

General Rules

The value expression and the subquery data types must be respectively comparable.

8.3.9.7 EXISTS Predicate

Specify a test for a non-empty set.

Format

<EXISTS predicate> ::= EXISTS <subquery>

Referenced section

Element	Section
<Subquery>	"Subquery"

8.3.10 Search Condition

Specify a condition containing the truth value true, false, or unknown, depending on the result of applying Boolean operators to specified conditions.

Format

<search condition> ::= <Boolean term> [OR <search condition>]

<Boolean term> ::= <Boolean factor> [AND <Boolean term>]

<Boolean factor> ::= [NOT] <Boolean primary>

<Boolean primary> ::= <predicate> | (<search condition>)

Referenced section

Element	Section
<predicate>	"Predicate"

8.3.11 Table Expression

Specify a table or a grouped table.

Format

<table expression> ::=

<FROM clause>

[<WHERE clause>]

[<GROUP BY clause>]

[<HAVING clause>]

8.3.11.1 FROM Clause

Specify a table derived from one or more named tables.

Format

<FROM clause> ::= FROM <table reference> [{,<table reference>}...]

<table reference> ::= <table name> [<correlation name>]

Referenced section

Element	Section
<table name>	"Table name"
<correlation name>	"Correlation name"

8.3.11.2 WHERE Clause

Specify a table derived by applying the search condition to the result of the preceding FROM clause.

Format

<WHERE clause> ::= WHERE <search condition>

Referenced section

Element	Section
<search condition>	"Search condition"

8.3.11.3 GROUP BY Clause

Specify a grouped table derived by applying the GROUP BY clause to the result specified by the FROM and WHERE clauses.

Format

<GROUP BY clause> ::= GROUP BY <column specification> [{,<column specification>}...]

Referenced section

Element	Section
<column specification>	"Column specification"

8.3.11.4 HAVING Clause

Specify the group selection criteria to apply to items resulting from the preceding GROUP BY or FROM clauses.

Format

<HAVING clause> ::= HAVING <search condition>

Referenced section

Element	Section
<search condition>	"Search condition"

8.3.12 Query Specification

Specify a table derived from the table expressions result.

Format

SELECT [ALL|DISTINCT] <select list> <table expression>

<select list> ::= *|<select sublist> [{,<select sublist>}...]

<select sublist> ::= <value expression>|<table name>.*|<correlation name>.*

Referenced section

Element	Section
<table expression>	"Table expression"
<FROM clause>	"FROM clause"
<WHERE clause>	"WHERE clause"
<value expression>	"Value expression"
<table name>	"Table name"
<correlation name>	"Correlation name"

8.3.13 Query Expression

Specify a table.

Format

<query expression> ::=

{<query specification> | (<query expression>) } |

<query expression> UNION [ALL] {<query specification> | (<query expression>) }

Referenced section

Element	Section
<query specification>	"Query specification"

8.3.14 Subquery

Specify a value, a row or a table derived from the table expression.

Format

<subquery> ::= (SELECT [ALL|DISTINCT] <select list> <table expression>)

Referenced section

Element	Section
<select list>	"Query specification"

Element	Section
<table expression>	"Table expression"

8.3.15 FOR Clause

Specify the number of times the SQL statement should be executed or the number of rows manipulated by the SQL statement.

Format

<FOR clause>::= FOR <host variable> | <value specification>

Syntax Rules

1. The value specified in the FOR clause must be an integer equal to or greater than 1. The maximum value is the maximum repeat count of the OCCURS clause defined in Appendix B, "System Quantitative Restrictions".
2. The host variable specified in the FOR clause must be a 9 digit signed binary integer data item.
3. The host variable specified in the FOR clause must be unique without qualification.
4. The FOR clause can only be specified in the following SQL statements containing a host variable with multiple rows specified or a host variable with a table specified:
   - SELECT statement
   - INSERT statement
   - FETCH statement (searched)
   - UPDATE statement (searched)
   - DELETE statement
   - EXECUTE statement

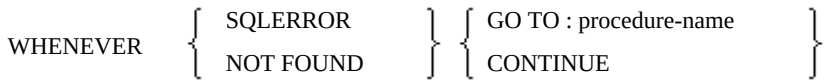
General Rules

1. The value in the FOR clause specified when fetching data from the database indicates the number of rows to be fetched. The value specified when manipulating data in the database indicates the number of times the SQL statement should be executed.
2. If 0 or a negative value is specified in the FOR clause, the SQL statement is not executed.
3. If the value specified in the FOR clause is greater than the occurrence number in a host variable with multiple rows specified or a host variable with a table specified, the SQL statement is not executed.
4. No FOR clause can be specified in a prepared statement.

8.4 Embedded Exception Declaration

Specify the action taken when an SQL statement causes an exception event.

Format



General Rules

1. If SQLERROR is specified, execution occurs except when SQLSTATE/SQLCODE indicates either normal termination or no data.

2. If NOT FOUND is specified, execution occurs when SQLSTATE/SQLCODE indicates no data.
3. The embedded exception declaration has an effect on all SQL statements encountered after the declaration in the source text until the end of the program or until another declaration appears for the same condition.
4. The embedded exception declaration scope is the program unit.

8.5 Data Manipulation without Using Cursor

The following statements manipulate data without using the cursor:

- SELECT statement
- DELETE statement (searched)
- INSERT statement
- UPDATE statement (searched)

8.5.1 SELECT Statement

The SELECT statement fetches the value from a single row in a table.

Format

[<FOR clause>]
SELECT [ALL|DISTINCT] <select list>
INTO <select target list>
<table expression>
<select target list> ::=
<target specification> [{,<target specification>}...]

Referenced section

Element	Section
<select list>	"Query specification"
<table expression>	"Table expression"
<target specification>	"Value specification and Target specification"

Syntax Rules

1. The select list degree and the target specification number in the selected target list must be equal.
2. The select list column and the target specification data types must be assignable.
3. When the FOR clause is specified, the target specifications in the select target list must all be host variables of multiple rows or host variables of a table.

General Rules

1. The SELECT statement search result is assigned to the host variable specified in the select target list. The assignment begins at the leftmost column of the select list and at the leftmost target specification of select target list, and continues in order.
2. When a host variable with multiple rows is specified in the select target list, assignment begins at the first row of the search result and at the first array element of the target specification and continues in order.

3. When a host variable with a table is specified in the select target list, assignment begins at the leftmost column of the select list and the subordinate items of the host variable. Assignment continues in order from the first row of the search result and first array element of the target specification.

8.5.2 DELETE Statement (Searched)

The DELETE statement (searched) deletes table rows meeting the search condition.

Format

[<FOR clause>]

DELETE FROM <table name> [WHERE <search condition>]

Referenced section

Element	Section
<table name>	"Table name"
<search condition>	"Search condition"

Syntax Rules

1. The table identified by the table name must be a read-write table.
2. When the FOR clause is specified, the host variables in the search condition must all be host variables with multiple rows specified.

General Rules

1. If a search condition is specified, rows meeting the search condition are deleted.
2. If a search condition is omitted, all rows in the table are deleted.
3. Search conditions are evaluated before rows in the table are deleted.
4. When a host variable with multiple rows is specified, the SQL statement is executed as many times as the occurrence of the host variable. When the FOR clause is specified, the SQL statement is executed as many times as the value specified in the FOR clause. For execution, values stored in an array are used in order from the first array element.

8.5.3 INSERT Statement

The INSERT statement inserts new rows into an existing table.

Format

[<FOR clause>]

INSERT INTO <table name> [(<inserted column list>)]

{VALUES (<insert value list>) | <query specification> }

<inserted column list> ::= <column name> [{,<column name>}...]

<insert value list> ::= <insert value> [{,<insert value>}...]

<insert value> ::= <value specification>|<dynamic parameter specification>|NULL

Referenced section

Element	Section
<query specification>	"Query specification"

Element	Section
<table name>	"Table name"
<column name>	"Column name"
<search condition>	"Search condition"
<value specification>	"Value specification and Target specification"
<dynamic parameter specification>	"Value specification and Target specification"

Syntax Rules

1. The column name in the insert value list must exist in the table identified by table name, and must be unique.
2. If an insert column list is omitted, the value is set in all table columns.
3. If an insert column list is specified, the column numbers in the insert column list must be equal to the query specification degree. If insert column list is omitted, the table degree must be equal to the query specification.
4. Every column in the query specification must be respectively assignable to the object table.
5. When the FOR clause is specified, the host variables with the insert value list or query specification must all be host variables of multiple rows or host variables of a table.

General Rules

1. The query specification result must not be empty.
2. When a host variable with multiple rows or a host variable with a table is specified, the SQL statement is executed as many times as the occurrence of the host variable with multiple rows or host variable with a table. When the FOR clause is specified, the SQL statement is executed as many times as the value specified in the FOR clause. For execution, values stored in an array are used in order from the first array element.

8.5.4 UPDATE Statement (Searched)

The UPDATE statement (searched) updates rows meeting the search condition.

Format

[<FOR clause>]

UPDATE <table name>

SET <set clause:searched> [{,<set clause:searched>}...]

[WHERE <search condition>]

<set clause:searched> ::= <object column:searched> = {<value expression>|NULL}

<object column:searched> ::= <column name>

Referenced section

Element	Section
<table name>	"Table name"
<search condition>	"Search condition"
<value expression>	"Value expression"
<column name>	"Column name"

Syntax Rules

1. The table identified by table name must be a read-write table.

2. The value expression in the SET clause must not include a set function specification.
3. The column name in the SET clause must be a column name existing in the object table.
4. The column name specified as the object column must be a column name existing in the object table.
5. In the SET clause, the value expression on the right side of the equal sign must be assignable to the column identified by column name on the left side of the equal sign.
6. When the FOR clause is specified, the host variables in the SET clause or search condition must all be host variables with multiple rows.

General Rules

1. If the WHERE clause is specified, rows meeting the search condition are updated.
2. If the WHERE clause is omitted, all rows in the table are updated.
3. The SET clause updates the row value to the value specified on the right side of the equal sign.
4. The column specification value included in the SET clause value expression indicates the value before the UPDATE statement (searched) update.
5. When a host variable with multiple rows is specified, the SQL statement is executed as many times as the occurrence of the host variable. When the FOR clause is specified, the SQL statement is executed as many times as the value specified in the FOR clause. For execution, values stored in an array are used in order from the first array element.

8.6 Data Manipulation Using the Cursor

The following statements manipulate data using the cursor:

- Declare cursor
- OPEN statement
- CLOSE statement
- FETCH statement
- DELETE statement (positioned)
- UPDATE statement (positioned)

8.6.1 Declare Cursor

Define a cursor.

Format

DECLARE <cursor name> CURSOR FOR <cursor specification>

<cursor specification> ::=

{<query expression> [<ORDER BY clause>]

|SELECT [ALL|DISTINCT] <select list>

<FROM clause> [<WHERE clause>]

[FOR UPDATE] }

<ORDER BY clause> ::= ORDER BY <sort specification> [{,<sort specification>}....]

<sort specification> ::= {<unsigned integer>|<column specification>} [ASC|DESC]

Referenced section

Element	Section
<cursor name>	"Cursor name"
<query expression>	"Query expression"
<select list>	"Query specification"
<FROM clause>	"FROM clause"
<WHERE clause>	"WHERE clause"
<column specification>	"Column specification"

Syntax Rules

1. The cursor name must be unique within the compilation unit.
2. If the ORDER BY clause is specified, rows in the derived table are ordered as described in the ORDER BY clause.
3. The sort specification in the ORDER BY clause must reference a column in the derived table. The format of the sort specification is as follows:
    a. If the sort specification contains a column specification, the sort specification must specify a derived table column name.
    b. If the sort specification contains an unsigned integer, the sort specification must specify a derived table column number. The unsigned integer must be greater than 0 and not greater than the number of columns in the derived table.
    c. Column specification and an unsigned integer can be mixed in multiple sort specifications.

General Rules

1. A cursor declaration must appear before any SQL statement using the cursor.
2. The derived table is a virtual table specified by a query expression. Once a derived table is defined by a cursor declaration, the table is valid within the compilation unit.
3. The cursor is in the closed state following the cursor definition.
4. The column name, data type, precision, scale, and length of a derived table are equal to the query expression values specified in the cursor declaration.
5. The ORDER BY clause effects are the following:
    a. If ASC is specified, rows are given in ascending order.
    b. If DESC is specified, rows are given in descending order.
    c. The sort priority among sort specifications is determined by the order in which they are specified.
    d. If sorted columns have the same value, they are returned in the same order they are encountered in the source table.
6. The DELETE statement (positioned) and UPDATE statement (positioned) cannot affect a read-only derived table.

8.6.2 OPEN Statement

The OPEN statement opens and validates a cursor.

Format

OPEN <cursor name>

Referenced section

Element	Section
<cursor name>	"Cursor name"

General Rules

1. The OPEN statement opens a cursor.
2. Only a closed cursor can be opened.
3. Following the opening of the cursor, the cursor is positioned just before the first row.

8.6.3 CLOSE Statement

The CLOSE statement closes and invalidates the cursor.

Format

CLOSE <cursor name>

Referenced section

Element	Section
<cursor name>	"Cursor name"

General Rules

1. Only an opened cursor can be closed.
2. A cursor in a closed state cannot be used again until it is reopened by an OPEN statement.

8.6.4 FETCH Statement

In [Win16] and [Linux], the FETCH statement positions the cursor at the specified row and fetches the values from the row.

In [Win32], [Winx64] and [.NET], the FETCH statement positions the cursor at the specified row and fetches the values

Format

[<FOR clause>]

FETCH [NEXT | PRIOR | FIRST | LAST] <cursor name> INTO <fetch target list>

<fetch target list> ::= <target specification> [{,<target specification>}...]

The NEXT and PRIOR functions are for use with [Win32], [Winx64] and [.NET].

The FIRST and LAST functions are for use with [Win32], [Winx64] and [.NET].

Referenced section

Element	Section
<cursor name>	"Cursor name"
<target specification>	"Value specification and Target specification"

Syntax Rules

1. The number of columns in the derived table must be equal to the number of host variables specified in fetch target list.
2. Each column in the derived table must be assignable to a host variable specified in fetch target list.

3. When the FOR clause is specified, the target specification in the fetch target list must all be host variables with multiple rows specified or host variables of a table.

General Rules

1. The cursor must be in the open state.
2. The row values are read from the leftmost column of the derived table, and are assigned to host variables in the order specified in the fetch target list.
3. When a host variable with multiple rows is specified in the fetch target list, the values read from the derived table are assigned in order from the first row of the derived table beginning at the first array element of the fetch target list.
4. When a host variable with a table specified is specified in the fetch target list, the values read from the derived table are associated with the subordinate items of the host variable, in order, from the leftmost column in the derived table. The values are further assigned in order from the first row of the derived table and first array element of the fetch target list.
5. In [Win32], [Winx64] and [.NET], NEXT and PRIOR specify the direction of the cursor.
    a. NEXT fetches the values from the row located after the specified row. NEXT is the default.
    b. PRIOR fetches the values from the row located before the specified row.
6. In [Win32], [Winx64] and [.NET], FIRST and LAST specify the location of the cursor.
    a. FIRST fetches the values after setting the cursor position to the first row in the derived table.
    b. LAST fetches the values after setting the cursor position to the last row in the derived table.

8.6.5 DELETE Statement (Positioned)

The DELETE statement (positioned) deletes one row specified by the cursor.

Format

```
DELETE FROM <table-name>
    WHERE CURRENT OF <cursor-name>
```

Referenced section

Element	Section
<table-name>	"Table name"
<cursor-name>	"Cursor name"

Syntax Rules

The table name identified by table-name must be the table deriving the cursor.

General Rules

The DELETE statement (positioned) deletes the row from the derived current row of the cursor.

8.6.6 UPDATE Statement (Positioned)

The UPDATE statement (positioned) updates the table row specified by the cursor.

Format

```
UPDATE <table-name>
    SET <set clause:positioned> [{,<set clause:positioned>}... ]
```

WHERE CURRENT OF <cursor-name>

<set clause:positioned> ::= <object column:positioned> = {<value-expression>|NULL}

<object column:positioned> ::= <column-name>

Referenced section

Element	Section
<table-name>	"Table name"
<cursor-name>	"Cursor name"
<value-expression>	"Value expression"
<column-name>	"Column name"

Syntax Rules

1. The table name identified by table-name must be the table deriving the cursor.
2. The following rules are applied to set clause:
    a. The column name in the SET clause must be a column name existing in the table deriving the cursor.
    b. The column name specified for SET clause must be unique.
    c. The value expression specified in SET clause must assignable to the associated column.

General Rules

1. The UPDATE statement (positioned) updates the value of the row at the derived current row of the cursor.
2. The row value is updated to the value specified on the right side of the equal sign.
3. The column specification value included in a value expression indicates the value before the UPDATE statement (positioned) update.

8.7 Dynamic SQL

This section describes dynamic SQL.

8.7.1 INTO Clause / USING Clause

The INTO clause/USING clause describes the dynamic SQL statement output variable and input parameter.

Format

{INTO|USING} {<target-specification> [{,<target specification>}...]

Referenced section

Element	Section
<target-specification>	"Value specification and Target specification"

8.7.2 PREPARE Statement

The PREPARE statement prepares an SQL statement for execution.

Format

PREPARE <SQL statement identifier> FROM <SQL statement variable>

<SQL statement variable> ::= <host variable name>

Referenced section

Element	Section
<SQL statement identifier>	"SQL statement identifier"
<host variable name>	"Value specification and Target specification"

Syntax Rules

The SQL statement variable data type must be either a fixed-length or variable-length character string of alphanumeric data item.

General Rules

1. EXEC SQL, END-EXEC, host variable, and FOR clause specification cannot be included in a dynamic SQL statement .
2. Instead of variable specification, the dynamic parameter specification can be included in the SQL statement variable contents.
3. The preparation for executing an SQL statement is completed by execution of the PREPARE statement after moving the SQL statement to the SQL statement variable. The statement prepared with the PREPARE statement is called a prepared statement.
4. If the content of a SQL statement variable is a dynamic SELECT statement, the cursor related to the SQL statement identifier must be in the closed state.

8.7.3 EXECUTE Statement

The EXECUTE statement associates input parameters and output targets with a prepared statement and executes the statement.

Format 1

EXECUTE <SQL statement identifier> [<INTO clause>] [<USING clause>]

Format 2

[<FOR clause>]

EXECUTE <SQL statement identifier> { <INTO clause> | <USING clause> }

Referenced section

Element	Section
<SQL statement identifier>	"SQL statement identifier"
<INTO clause>	"INTO clause/USING clause"
<USING clause>	"INTO clause/USING clause"

Syntax Rules

A host variable with multiple rows or a host variable with a table must be specified in the INTO or USING clause of Format 2.

General Rules

1. The prepared statement associated with an SQL statement identifier must be previously prepared within the same compilation unit by the PREPARE statement.
2. The following six prepared statements can be executed by the EXECUTE statement in Format 1:
   - SELECT statement
   - DELETE statement (searched)
   - DELETE statement (positioned)
   - INSERT statement
   - UPDATE statement (searched)
   - UPDATE statement (positioned)
3. The following four prepared statements can be executed by the EXECUTE statement in Format 2:
   - SELECT statement
   - DELETE statement (searched)
   - INSERT statement
   - UPDATE statement (searched)
4. If a dynamic parameter is specified in a prepared statement, the USING clause must be specified.
5. Target specification in a USING clause must be initialized with a value in advance of its use.
6. If a prepared statement is a dynamic single row SELECT statement, the INTO clause must be specified.
7. The EXECUTE statement associates the input parameter specified in the USING clause with the dynamic parameter value and executes the prepared statement. If the prepared statement is a dynamic single row SELECT statement, the execution result value is moved to the INTO clause host variable.

8.7.4 EXECUTE IMMEDIATE Statement

The EXECUTE IMMEDIATE statement dynamically prepares and executes a prepared statement.

Format

EXECUTE IMMEDIATE <SQL statement variable>

Referenced section

Element	Section
<SQL statement variable>	"PREPARE statement"

Syntax Rules

The SQL statement variable data type must be a fixed-length or variable-length character string of alphanumeric data item.

General Rules

1. The following five SQL statements can be specified as the SQL statement variable contents:
   - DELETE statement (searched)
   - DELETE statement (positioned)
   - INSERT statement
   - UPDATE statement (searched)

- UPDATE statement (positioned)

2. EXEC SQL, END-EXEC, host variable specification, dynamic parameter, and FOR clause specification cannot be included in the contents of the SQL statement variable.

8.7.5 Dynamic SELECT Statement

To code a dynamic SELECT statement, follow the formats described in "Query Specification" and "Subquery" topics earlier in this chapter.

The dynamic SELECT statement can include dynamic parameter specifications.

8.7.6 Dynamic Declare Cursor

Declare the cursor associated with the SQL statement identifier prepared as a dynamic SELECT statement.

Format

DECLARE <cursor name> CURSOR FOR <SQL statement identifier>

Referenced section

Element	Section
<cursor name>	"Cursor name"
<SQL statement identifier>	"SQL statement identifier"

Syntax Rules

1. The cursor name must be unique within a compilation unit.
2. The SQL statement identifier specified in the dynamic cursor declaration must be prepared with the PREPARE statement within the same compilation unit.

General Rules

1. The prepared statement associated with the specified dynamic cursor declaration SQL statement identifier must be a cursor specification (dynamic SELECT statement). The cursor specification related to the SQL statement identifier can include dynamic parameter specification.
2. The cursor defined by a dynamic cursor declaration can be used in the same way as a cursor defined by a cursor declaration.

8.7.7 Dynamic OPEN Statement

The dynamic OPEN statement associates input parameters with the cursor specification, and opens the cursor.

Format

OPEN <cursor name> [<USING clause>]

Referenced section

Element	Section
<cursor name>	"Cursor name"
<USING clause>	"INTO clause/USING clause"

Syntax Rules

The OPEN statement syntax rules can be applied to the dynamic OPEN statement by replacing a cursor declaration with a dynamic cursor declaration and replacing an OPEN statement with a dynamic OPEN statement.

General Rules

1. The prepared statement associated with a cursor name must be a cursor specification.
2. If the cursor specification associated with a cursor name includes a dynamic parameter specification, a USING clause must be specified in the dynamic OPEN statement.
3. The dynamic parameter specification value in the prepared statement is set in the USING clause target specification.
4. The OPEN statement general rules can be applied to the dynamic OPEN statement by replacing a cursor declaration with a dynamic cursor declaration.

8.7.8 Dynamic CLOSE Statement

The dynamic CLOSE statement closes the cursor.

Format

CLOSE <cursor name>

Referenced section

Element	Section
<cursor name>	"Cursor name"

Syntax Rules

The CLOSE statement syntax rules can be applied to the dynamic CLOSE statement by replacing a cursor declaration with a dynamic cursor declaration and replacing a CLOSE statement with a dynamic CLOSE statement.

General Rules

The CLOSE statement general rules can be applied to the dynamic CLOSE statement.

8.7.9 Dynamic FETCH Statement

The dynamic FETCH statement fetches a row for a cursor declared with a dynamic cursor declaration.

Format

[<FOR clause>]

FETCH <cursor name> <INTO clause>

Referenced section

Element	Section
<cursor name>	"Cursor name"
<INTO clause>	"INTO clause/USING clause"

Syntax Rules

The FETCH statement syntax rules can be applied to the dynamic FETCH statement by replacing a cursor declaration with a dynamic cursor declaration.

General Rules

The FETCH statement general rules can be applied to the dynamic FETCH statement by replacing a cursor declaration with a dynamic cursor declaration.

8.7.10 Dynamic DELETE Statement (Positioned)

The dynamic DELETE statement (positioned) deletes a row of a table.

Format

DELETE FROM <table name>

WHERE CURRENT OF <cursor name>

Referenced section

Element	Section
<table name>	"Table name"
<cursor name>	"Cursor name"

Syntax Rules

The DELETE statement (positioned) syntax rules can be applied to the dynamic DELETE statement (positioned) by replacing a cursor declarative with a dynamic cursor declarative and replacing a DELETE statement (positioned) with a dynamic DELETE statement (positioned).

General Rules

The DELETE statement (positioned) general rules can be applied to the dynamic DELETE statement (positioned) by replacing a DELETE statement (positioned) with a dynamic DELETE statement (positioned).

8.7.11 Dynamic UPDATE Statement (Positioned)

The dynamic UPDATE statement (positioned) updates a row of a table.

Format

UPDATE <table name>

SET <set clause:positioned> [{,<set clause:positioned>}...]

WHERE CURRENT OF <cursor name>

Referenced section

Element	Section
<table name>	"Table name"
<set clause:positioned>	"UPDATE Statement (positioned)"
<cursor name>	"Cursor name"

Syntax Rules

The UPDATE statement (positioned) syntax rules can be applied to the dynamic UPDATE statement (positioned) by replacing a cursor declaration with a dynamic cursor declaration and replacing an UPDATE statement (positioned) with a dynamic UPDATE statement (positioned).

General Rules

The UPDATE statement (positioned) general rules can be applied to the dynamic UPDATE statement (positioned) by replacing an UPDATE statement (positioned) with a dynamic UPDATE statement (positioned).

8.8 Transaction Management

The session control statements are:

- COMMIT statement
- ROLLBACK statement

8.8.1 COMMIT Statement

Terminates the current transaction with a commit.

Format

COMMIT [WORK]

General Rules

1. The current transaction is terminated.
2. Any changes made by the current transaction are reflected in the database.

8.8.2 ROLLBACK Statement

Terminates the current transaction with a rollback.

Format

ROLLBACK [WORK]

General Rules

1. The current transaction is terminated.
2. Any changes made by the current transaction are canceled.

8.9 Connection Management

The connection control statements are:

- CONNECT statement
- SET CONNECTION statement
- DISCONNECT statement

8.9.1 CONNECT Statement

Establishes an SQL connection.

Format

CONNECT TO <connection target>

<connection target> ::=

{<server name> [AS <connection name>]

[USER <user name>]

|DEFAULT}

<server name> ::= See the General rules.

<connection name> ::= See the General rules.

<user name> ::= See the General rules.

General Rules

1. The server name, connection name, and user name must each be a character string literal or a host variable whose data type is fixed-length character string.
2. The server name is the same name specified as the server in the operating environment file.
3. The maximum length of a server name is 32 bytes. However, this rule may be restricted by the users DBMS.
4. The connection name identifies the connection.
5. The maximum length of the connection name is 18 bytes. However, this rule may be restricted by the DBMS being used.
6. If the connection name is omitted, the server name is assumed to be the connection name.
7. A user name consists of a user-ID and a password, separated by a slash (/). The user-ID and a password must be valid on the server.
8. The maximum length of a user-ID and a password is 32 bytes each. However, this rule may be restricted by the DBMS being used.
9. Spaces before and after a server name, a connection name, a user-ID, or a password are ignored.
10. If USER is omitted, the default connection information in the operating environment file is used.
11. If DEFAULT is specified, the default connection information in the operating environment file is used.
12. Execution of a CONNECT statement establishes a connection to a server.
13. If two or more connections are established, the last connection is the current connection.
14. Two or more connections cannot have the same connection name within the scope of the connection. Also, two or more connections cannot specify DEFAULT.

8.9.2 SET CONNECTION Statement

Selects one connection from the available connections.

Format

SET CONNECTION <connection object>

<connection object> ::= <connection name>|DEFAULT

<connection name> ::= See the General rules.

General Rules

1. A connection name must be a character string literal or a host variable whose data type is a fixed-length character string.
2. The maximum length of a connection name is 18 bytes. However, this rule may be restricted by the DBMS being used.
3. A connection name specifies the available connection.
4. If DEFAULT is specified, the default connection was established previously.

8.9.3 DISCONNECT Statement

Terminates a SQL connection.

Format

DISCONNECT <disconnection object>

<disconnection object> ::= <connection name>|DEFAULT|ALL|CURRENT

<connection name> ::= See the General rules.

General Rules

1. A host variable with a fixed string type or a character literal is specified as a connection name.
2. The maximum length of connection name is 18 bytes. This rule may, however, be restricted by the DBMS used.
3. If a connection name is specified, only that connection is disconnected.
4. If DEFAULT is specified, only the default connection is disconnected.
5. If ALL is specified, all existing connections are disconnected.
6. If CURRENT is specified, only the current connection is disconnected.
7. The current connection becomes "none" if the current connection is specified by a DISCONNECT statement.
8. Any active transaction must be ended when a DISCONNECT statement is performed.

8.10 Stored Procedure

This section describes the following statement:

- CALL statement

8.10.1 The CALL Statement

This statement calls a stored procedure stored in a server.

This function is specific to [Win32], [Winx64], [Linux] and [.NET].

Format

CALL <stored-procedure-name>([<argument list>])

<argument list>::=<argument>[{,<argument>}...]

<argument>::=<value specification>

Refer to:

"Stored Procedure Names" topic earlier in this chapter

<value specification> see "Value Specification and Target Specification" topic earlier in this chapter

General Rules

1. The number of arguments specified should be the same as the number of parameter definitions in the stored procedure.
2. The order of specified arguments must be the same as the order of corresponding parameters in the definition of a stored procedure.
3. In [Win32], [Winx64] and [.NET], SQLERRD(1) displays the return value of the stored procedure definition. SQLERRD(1) is the first element of SQLERRD.
4. A literal that is specified as an argument can only be specified as an input parameter to the stored procedure definition.

F

Chapter 9 Communication Database

The communication database function transmits and receives data via the client server model communication database. The communication database function is specific to [Win16].

General Overview

In a client server model, the client sends unprocessed data to the server application with a request to process the data. The server application honors the request and returns the processing results to the client. In this communication model, the server application and client do not directly communicate with each other but send and receive data via the communication database.

When client data is sent, the client transmits data instructions directed to the communication database. The server application issues a data receive instruction directed to the communication database, and receives the data transmitted from the client.

When server data is sent from the application, the server application transmits data instructions directed to the communication database. The client issues a data receive instruction directed to the communication database, and receives data transmitted from the server application.

The communication database allows the user to communicate with a location destination, irrespective of the internal processing taking place, the server application, or where the actual client is.

The communication database function is specific to [Win16]. Both server and client must be running the PowerAIM products and its communication software to use the communication database.

Communication Database

The communication database is a virtual database. However, because the communication database is accessed using SQL, it functions as a real database. The user sends or receives data via the communication database.

The communication database provides the necessary services and tables for linkage between the server application and the client.

Services

Service information is held in tables available for the user.

When the client specifies a service, the corresponding server application is activated so the client can communicate with it.

A service is associated one to one with a server application.

Tables

Tables define the communication data attributes. Tables are open to users sending or receiving data conforming to table definitions.

There are input tables and output tables. An input table defines the data attributes received from the client by the server application. An output table defines the data attributes sent to the client by the server application.

Embedded DCSQL

Embedded DCSQL is designed to communicate within application programs using the communication database. DCSQL is part of SQL used for accessing the communication database.

"EXEC DCSQL" and "END-EXEC" pair encloses an SQL statement description in embedded DCSQL in the COBOL program.

Host Variable

Host variables are data items used for sending and receiving data between server applications and clients.

For further details regarding host variables, see the topic titled "Host Variable Names" of the "Basic Elements of Embedded DCSQL" section below and the topic titled "Host Variable Definitions" of the "Embedded DCSQL" section later in this chapter.

9.1 Basic Elements of Embedded DCSQL

This section explains the basic elements of embedded DCSQL in a COBOL program.

9.1.1 Available Characters

The following six characters types can be used for embedded DCSQL:

- Digits
- Alphabetic characters
- Alphanumeric characters
- Special characters
- Extended characters
- National characters

Digits

The characters: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9

Alphabetic Characters

Uppercase and lowercase alphabetic characters are the following:

- Uppercase alphabetic characters (A, B, C, ...,Z)
- Lowercase alphabetic characters (a, b, c, ...,z)

Alphanumeric Characters

An alphanumeric character is a set of alphabetic and numeric characters

Special Characters

The following 16 special characters are available:

.	<	(	+	*	)	;	-
/	,	%	_	>	?	:	"

Extended Characters

The following 3 extended characters are available:

@, \, #

National Characters

National character is available.

9.1.2 Quotation Marks, Key Words, and Separators

In embedded DCSQL, an apostrophe (') is used as the quotation mark. To write a quotation mark in a character string literal, specify two quotation marks in succession. Within a character string literal, the compiler uses a single apostrophe in terms of value and length.

The rules for key words and separators in embedded DCSQL statements follow the rules for embedded SQL.

9.1.3 Communication Database Names

The communication database name must conform to the following rules:

- 18 or less alphanumeric characters
- 8 or less national characters

9.1.4 Service Names

The service name must conform to the following rules:

- 18 or less alphanumeric characters
- 18 or less national characters

9.1.5 Table Names

Table names must conform to the following rules:

- 18 or less alphanumeric characters
- 18 or less national characters

9.1.6 Host Variable Names

Host variable names are names the user assigns to host variables. Host variable names can be specified for any sections permitting the specification of identifiers or data-names in embedded DCSQL statements of the PROCEDURE DIVISION and COBOL programs.

A host variable name must be a list of 18 or less alphanumeric characters or national characters. The configuration characters must follow the user defined words rules in COBOL.

9.1.7 Literals

The following literals can be specified in embedded DCSQL:

- Character string literal
- National character string literal
- Exact numeric literal
- Approximate literal

9.1.7.1 Character String Literal

A character string literal is a character string starting and ending with a quotation mark. The character string literal specifies character data.

Format

'{character-1}...'

Syntax Rules

1. To include a quotation mark in a character string literal, specify two quotation marks in succession. Within a character string literal, the compiler uses a single apostrophe in terms of value and length.
2. The character string literal data type is character string.
3. The character string literal length conforms to the host variable length rules whose data type is character string.

General Rules

The character string literal value is the character series the literal contains.

9.1.7.2 National Character String Literal

A national character string literal is a national character string starting with the character N, a quotation mark, a list of national characters, and ends with a quotation mark. This literal specifies each national character.

Format

N'{national-character-1}...'

Syntax Rules

1. The national character string literal data type is national character string. The national character string literal length is the number of national characters included in the literal.
2. The national character string literal length conforms to the host variable length rules whose data type is national character string type.

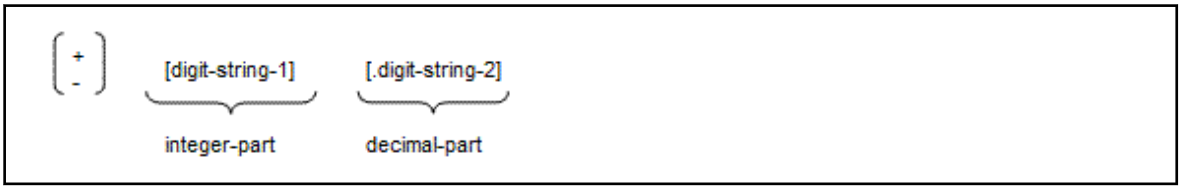
General Rules

The national character string literal value is the national characters series the literal contains.

9.1.7.3 Exact Numeric Literal

An exact numeric literal is a character string consisting of digits 0 to 9 and a decimal point. A positive or negative sign can be specified for an exact numeric literal.

Format



Syntax Rules

1. The exact numeric literal data type is an exact number. The number of digits it contains defines the precision of an exact numeric literal. The scale of an exact numeric literal is the number of digits to the right of the decimal point.
2. The precision and scale of an exact numeric literal conform to the host variables rules for precision and scale where the data type is an exact numeric.

General Rules

1. An exact numeric literal represents fixed-point numeric data.
2. The exact numeric literal numeric value is the value derived from normal numeric interpretation of signed positional decimal notation.

9.1.7.4 Approximate Literal

An approximate literal is a character string consisting of digits 0 to 9, a decimal point, the character E, and a character string consisting of digits 0 to 9. A positive or negative sign can be specified for an approximate literal.

Format

[+|-]mantissa E exponent

Syntax Rules

1. The approximate literal data type is approximate. The approximate precision is the precision of its mantissa.

2. Specify the mantissa in exact numeric literal format.
3. Specify the exponent in the following format:

```
[+|-]{number}...
```

4. The precision and scale of an approximate literal conform to the host variable rules for precision and scale where the data type is the approximate.

General Rules

1. An approximate represents floating-point numeric data.
2. The approximate literal value is the product of the exact numeric expressed by the mantissa and the power of ten expressed by the exponent.

9.2 Reference Format of Embedded DCSQL

This section explains the reference format of embedded DCSQL described in COBOL programs.

9.2.1 Overall Rules for Description

1. The embedded DCSQL begin declare, embedded DCSQL end declare, and embedded DCSQL statement must be enclosed by the DCSQL prefix (EXEC DCSQL) and the DCSQL terminator (END-EXEC).
2. The embedded DCSQL begin declare, embedded DCSQL end declare, and embedded DCSQL statements must all be described in Area B.

9.2.2 Continuation of Line

The rules of continuation (continuation of line) of an embedded DCSQL begin declare, embedded DCSQL end declare, and embedded DCSQL statement conform to the format rules of COBOL.

9.2.3 COBOL Comment Line and In-line Comment

COBOL comment lines, debugging lines, and blank lines can be included in embedded DCSQL; however, in-line comments cannot.

9.2.4 Comments in Embedded DCSQL

A comment in embedded DCSQL begins with two consecutive hyphens (--) and ends at the character position immediately to the left of Margin R. A comment in embedded DCSQL may begin at any position in Area A or Area B of embedded DCSQL statements, however, a space or a comma must immediately precede the DCSQL comment. A comment in embedded DCSQL cannot be continued. To continue a word, literal, or PICTURE clause character string immediately preceding embedded DCSQL comments, place a hyphen in the indicator area.

9.3 Embedded DCSQL

Embedded DCSQL is designed to communicate within application programs using the communication database. This section describes the formats and rules for the embedded DCSQL declarative section and the embedded DCSQL statement.

9.3.1 Data Division

In the DATA DIVISION, variables used in the embedded DCSQL statement are defined.

9.3.1.1 Embedded DCSQL Declare Section

In the embedded DCSQL declare section, define the following:

- Host variables used for communication between the server and client.
- Variables used for receiving the processing result status reported by the system
- Variables used for receiving system messages

Format

EXEC DCSQL

BEGIN DECLARE SECTION END-EXEC.

[{host variable definition}...]

EXEC DCSQL

END DECLARE SECTION END-EXEC.

General Rules

1. An embedded DCSQL declare section can be included in the WORKING-STORAGE SECTION or the LINKAGE SECTION of the DATA DIVISION.
2. An embedded DCSQL begin declare (EXEC DCSQL BEGIN DECLARE SECTION END-EXEC) and an embedded DCSQL end declare (EXEC DCSQL END DECLARE SECTION END-EXEC) cannot be omitted.
3. An embedded DCSQL BEGIN DECLARE must be paired with an embedded DCSQL END DECLARE. They may not be nested.
4. An embedded DCSQL declare section may not be included in an embedded SQL declare section. An embedded SQL declare section may not be included in an embedded DCSQL declare section.

9.3.1.2 Host Variable Definitions

Host variable definitions specify the host variable characteristics.

When a floating-point data item is defined as a host variable, COMP-1 or COMP-2 is specified in the USAGE clause.

Format

level-numbervariable-name

[IS EXTERNAL]

[IS GLOBAL]

[{ PICTURE / PIC } IS character-string]

[USAGE IS] { BINARY / COMPUTATIONAL / COMP / COMP-1 / COMP-2 / DISPLAY / PACKED-DECIMAL }

[[SIGN IS]

LEADING SEPARATE CHARACTER]

[VALUE IS literal-1].

Syntax Rules

1. A host variable must be declared as an elementary item with level number 01 or 77.
2. A host variable corresponding to an item defined in a table as data type CHARACTER VARYING or NATIONAL CHARACTER VARYING must be declared as a level-number 01 group item. In this case, items directly belonging to level-number 01 must be level-number 49 elementary items.
3. The syntax rules of each clause must conform to standard COBOL syntax rules.

General Rules

1. A PICTURE clause can only be specified at an elementary item level.
2. The general rule of each clause must conform to standard COBOL general rules.
3. When a host variable is specified, it must be one of the following elementary items:
   a. Signed binary data item of 4 or 9 digits
   b. Signed packed decimal data item of 18 or less digits
   c. Signed zoned decimal data item of 18 or less digits (SIGN clause must be specified.)
   d. Alphanumeric data item
   e. National data item
   f. Single-precision internal floating-point data item
   g. Double-precision internal floating-point data item
4. Host variables corresponding to an item defined in a table as data type CHARACTER VARYING must the following:

```
01  host-variable-name-1.
  49  host-variable-name-2 PIC S9(9) BINARY.
  49  host-variable-name-3 PIC X(n).
```

5. Host variables corresponding to an item defined in a table as data type NATIONAL CHARACTER VARYING must the following:

```
01  host-variable-name-1.
  49  host-variable-name-2  PIC S9(9)  BINARY.
  49  host-variable-name-3  PIC N(n).
```

6. A host variable cannot be a filler.

9.3.1.3 Referencing Host Variables

Host variables can be referenced either in embedded DCSQL statements or in general COBOL statements. If a host variable is referenced in an embedded DCSQL statement, a colon (:) must be added in front of the host variable name. For example, if host variable A is referenced, it must be declared as :A in the embedded DCSQL statement. If a host variable is referenced in a general COBOL statement, do not add the colon.

9.3.1.4 DCSQLSTATE

When a DCSQL statement is executed, information on successful or failed execution is coded and moved to DCSQLSTATE. The application program references DCSQLSTATE to determine the execution state of the DCSQL statement. The application program is allowed to alter its processing based on this information.

DCSQLSTATE must be defined in an embedded DCSQL declare section as a 5 character alphanumeric data item with a variable name of DCSQLSTATE.

Refer to "PowerAIM/CL Software Development Kit for Windows On-Line Manual" for the details of the DCSQLSTATE values.

9.3.1.5 DCSQLMSG

When the DCSQL statement is executed, a message indicating a successful or failed execution is stored in DCSQLMSG. The user can display or print the message stored in DCSQLMSG using an output statement.

To use DCSQLMSG, an alphanumeric data item with a variable name of DCSQLMSG must be defined in an embedded DCSQL declarative section. The maximum length of DCSQLMSG is 255 bytes.

9.3.2 Procedure Division

The statements defining the connection or the disconnection to the database, the transaction management, and the transmission and receiving of the data are described in the PROCEDURE DIVISION.

Refer to "PowerAIM/CL Software Development Kit for Windows On-Line Manual" for the details of the formats and coding rules.

F

Chapter 10 Micro Focus Native Functions

This chapter describes the functions and formats native to the Micro Focus implementation of COBOL.

10.1 Named Literal

A named literal is a mnemonic given to a literal to refer to allow reference to a literal by name.

Note

This is a function specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

10.1.1 User-defined Words

User-defined words contain the named literal in addition to those described in the topic "User-defined Words" in the "Basic Overview of Language" section of Chapter 1, "General Rules".

The Named Literal

The named literal is defined by level number 78 in the DATA DIVISION. The named literal may be used in place of "literal-n" or "integer-n" as shown in the COBOL syntax diagrams except for the integers indicating the number of occurrences of a PICTURE clause and the following.

- In a report description entry and report group description entries that are specific to the Report writer module
- In a concatenation expression
- In a COPY statement
- In an ENTRY statement

10.1.2 Scope of Name

A named literal can be referenced by a statement that is physically defined after the definition of the named literal, or in a statement of a program contained within the program in which the named literal is defined.

However, the named literal must not have the same name as any other user-defined words in the same compilation unit.

10.1.3 Data Description Entry

The data description entry is explained here.

Format

```
78 named-literal-1 VALUE IS { literal-1                  } .
                            { LENGTH OF identifier-1     }
```

Syntax Rules

1. A named literal can be defined in the following:
   - FILE SECTION
   - WORK-STORAGE SECTION
   - SCREEN SECTION
   - LITERAL SECTION
   - BASED-STORAGE SECTION

- LINKAGE SECTION
- LOCAL-STORAGE SECTION ([Winx64], [Linux64] and [.NET] only)

2. Level number 78 data items must be defined at the beginning of the sections mentioned in rule 1 above, and before other data description entries.
3. Literal-1 must not be a symbolic-character or symbolic-constant.
4. Identifier-1 must not have a subscript or reference modifier.
5. An internal Boolean data item must not be specified for an identifier-1.

General Rules

1. Named-literal-1 is the name given to the value of literal-1.
2. Characteristics of the named literal-1 are determined by the contents and type of literal-1.
3. When LENGTH is specified, it is assumed that the length of the memory area assigned to identifier-1 has already been specified in a prior data description entry.
4. Identifier-1 must be defined by a data description entry that is completely defined physically before this data description entry.
5. When a variable occurrence data item is specified as identifier-1, named-literal-1 is defined as if the value of the data item specified in the DEPENDING ON phrase of the OCCURS clause has been set to the maximum number of occurrences of the OCCURS clause.

10.2 Hexadecimal Numeric Literal

A hexadecimal numeric literal is a hexadecimal character string whose value represents a numeric value.

This support is specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].

Format

H"hexadecimal-character-string-1 ..."

1. The characters in hexadecimal-character-string-1 must be delimited by the separator H" at the left and by a double quotation mark at the right.
2. Hexadecimal-character-string-1 must consist of characters between 0 and 9 or A to F.
3. The number of characters in hexadecimal-character-string-1 must be an even number between 2 and
4. 16.
5. Hexadecimal-character-string-1 must be represented in big-endian format.
6. The hexadecimal numeric literal category is numeric.
7. A hexadecimal numeric literal can be specified wherever "literal-n" is indicated in a syntax diagram, and where a numeric literal is permitted.
8. The hexadecimal numeric literal cannot represent a decimal fraction or a negative number.
9. The hexadecimal numeric literal is aligned according to the standard alignment rules.

10.3 Screen Functions

This section describes the functions to manipulate screens offered by the Micro Focus implementation of COBOL.

Note

The Screen functions cannot be used in [Linux], [LinuxIPF], [Linux64] and [.NET].

10.3.1 Environment Division

This section describes the SPECIAL-NAMES paragraph.

10.3.1.1 Special-names Paragraph

The SPECIAL-NAMES paragraph defines symbolic-constants, symbolic-characters, alphabet-names, character sets, collating sequences, and associates function-names with mnemonic-names.

The CONSOLE IS CRT clause can be specified to the screen handling module.

Format

SPECIAL-NAMES.

[CONSOLE IS CRT clause]

General Rules

All clauses in the SPECIAL-NAMES paragraph apply to the program explicitly or implicitly specifying them and their nested programs.

10.3.1.1.1 CONSOLE IS CRT Clause

The CONSOLE IS CRT clause defines the implicitly-specified input-output destination.

Format

CONSOLE IS CRT

The CONSOLE IS CRT clause regards operands in the ACCEPT statement where the FROM clause is not specified, and operands in the DISPLAY statement where the UPON clause is not specified, as screen items.

10.3.2 Data Division

This section describes the screen data description entries.

10.3.2.1 Screen Data Description Entries

Format 1

Defines a group screen item

level-number [data-name-1 | FILLER]

[AUTO clause]

[BACKGROUND-COLOR clause]

[BLANK SCREEN clause]

[BLINK clause]

[CONTROL clause]

[FOREGROUND-COLOR clause]

[FULL clause]
[GRID clause]
[HIGHLIGHT clause]
[LOWLIGHT clause]
[LEFTLINE clause]
[OCCURS clause]
[OVERLINE clause]
[PROMPT clause]
[REQUIRED clause]
[REVERSE-VIDEO clause]
[SECURE clause]
[SIGN clause]
[UNDERLINE clause]
[USAGE clause]
[ZERO-FILL clause].

Format 2

Defines literal items

level-number [data-name-1]
[FILLER]
[BACKGROUND-COLOR clause]
[BELL clause]
[BLANK LINE clause]
[BLANK SCREEN clause]
[BLINK clause]
[COLUMN NUMBER clause]
[ERASE clause]
[FOREGROUND-COLOR clause]
[GRID clause]
[HIGHLIGHT clause]
[LOWLIGHT clause]
[LEFTLINE clause]
[LINE NUMBER clause]
[OCCURS clause]
[OVERLINE clause]
[REVERSE-VIDEO clause]
[SIZE clause]
[UNDERLINE clause]
VALUE clause.

Format 3

Defines input items, output items, or update items

```
level-number [ data-name-1 ]
             [ FILLER      ]
             [AUTO clause]
             [BACKGROUND-COLOR clause]
             [BELL clause]
             [ BLANK LINE clause   ]
             [ BLANK SCREEN clause ]
             [BLANK WHEN ZERO clause]
             [BLINK clause]
             [COLUMN NUMBER clause]
             [ERASE clause]
             [FOREGROUND-COLOR clause]
             [FULL clause]
             [GRID clause]
             [ HIGHLIGHT clause ]
             [ LOWLIGHT clause  ]
             [JUSTIFIED clause]
             [LEFTLINE clause]
             [LINE NUMBER clause]
             [OCCURS clause]
             [OVERLINE clause]
             [PICTURE clause]
             [PROMPT clause]
             [REQUIRED clause]
             [REVERSE-VIDEO clause]
             [SECURE clause]
             [SIGN clause]
             [SIZE clause]
             [UNDERLINE clause]
             [USAGE clause]
             [ZERO-FILL clause].
```

Syntax Rules

See the section titled "Screen Data Description Entry" of Chapter 5, the "Data Division" for the syntax rules.

General Rules

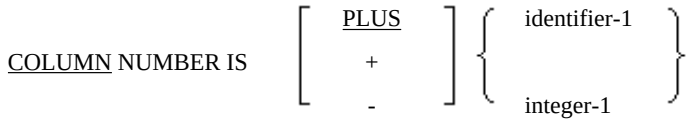
1. Use the PICTURE clause to classify an input item, output item, or update item.
2. When the SECURE clause is specified in a group screen item, the clause applies to all input items belonging to that group screen item.

3. When one of the following clauses is specified in a group screen item, the clause applies to all input items and update items belonging to the specified group screen item:
   - AUTO clause
   - FULL clause
   - REQUIRED clause
   - PROMPT clause
   - ZERO-FILL clause
4. Do not specify overlapping areas on the screen when using the LINE NUMBER and COLUMN NUMBER clauses. Also, do not specify areas exceeding the physical screen area.
5. The JUSTIFIED and the SIZE clauses cannot both be specified in the description of a screen item.
6. When the JUSTIFIED clause and the PROMPT clause are both specified, the PROMPT clause is invalidated.

10.3.2.1.1 COLUMN NUMBER Clause

The COLUMN NUMBER clause specifies the rows where screen items are arrayed. + and - can be used to specify relative row numbers.

Format



Syntax Rules

The format and rules for + and - conform to PLUS. See the topic titled "COLUMN NUMBER clause" in the "Screen Data Description Entry" of Chapter 5, the "Data Division" for syntax and general rules.

10.3.2.1.2 CONTROL Clause

Function

The CONTROL clause specifies the display attribute and behavior at execution time.

General Format

Format

CONTROL IS data-name-1

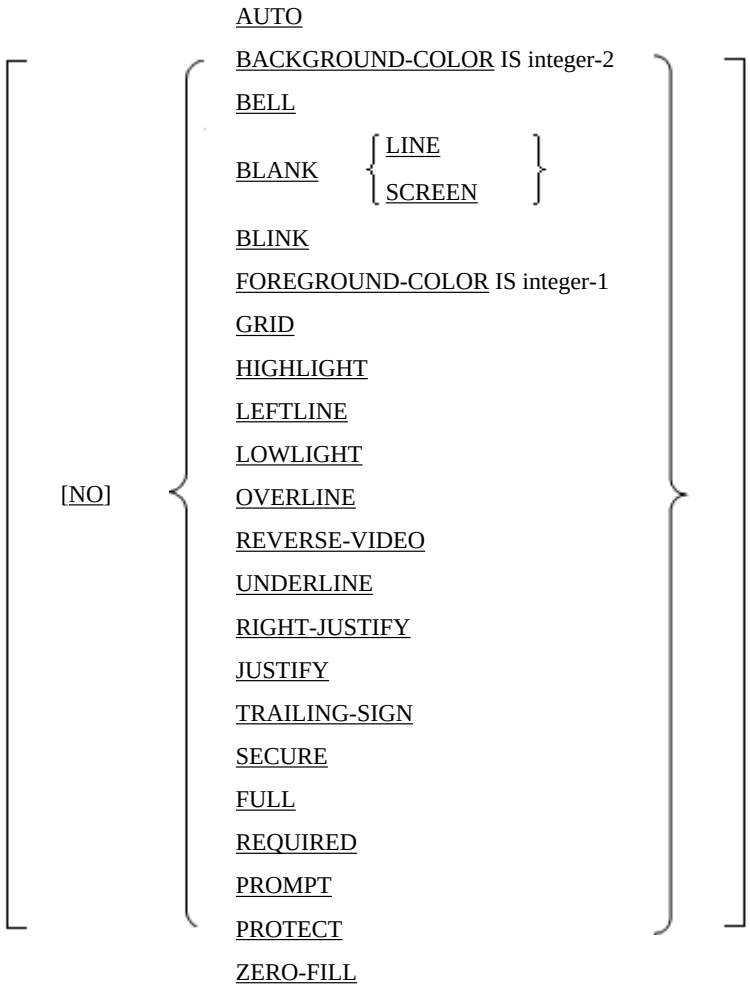
Syntax Rules

1. The CONTROL clause can be specified for any screen item.
2. data-name-1 must be an alphanumeric data item.

General Rules

1. When the CONTROL clause is specified in a group item, the display attribute and behavior specified in data-name-1 of the CONTROL clause apply to all elementary items contained in the group item.

   The following clauses can be specified in data-name-1:



2. If blank is specified in data-name-1 of the CONTROL clause, the display attribute and behavior specified in the screen item are used.
3. For information on the clauses that can be specified in date-name-1 of the CONTROL clause, see the descriptions of each clause.
4. The PROTECT clause must not be specified in an input item.

10.3.2.1.3 GRID Clause

The GRID clause specifies a vertical line appended to the left of each character in a screen item.

Format

GRID

Syntax Rules

1. The GRID clause can be specified for any screen item.
2. When the GRID clause is specified at a group item level, it applies to all elementary items belonging to that group item.

General Rules

When the GRID clause is specified for a screen item, a vertical line is appended to the left of each character in the displayed screen item.

10.3.2.1.4 LEFTLINE Clause

The LEFTLINE clause specifies a vertical line appended to the left of the character at the high order end of a screen item.

Format

LEFTLINE

Syntax Rules

1. The LEFTLINE clause can be specified for any screen item.
2. When the LEFTLINE clause is specified at a group item level, it applies to all elementary items belonging to that group item.

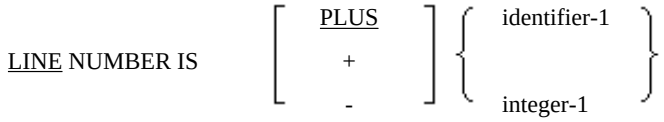
General Rules

When the LEFTLINE clause is specified for a screen item, a vertical line is appended to the left of the character at the high order end of the displayed screen item.

10.3.2.1.5 LINE NUMBER Clause

The LINE NUMBER clause specifies lines where screen items are arrayed. + and - can be used to specify relative line numbers.

Format



Syntax Rules

The format and rules for + and - conform to the PLUS clause. See the topic titled "LINE NUMBER clause" of the "Screen Data Description Entry" section of Chapter 5, the "Data Division" for syntax and general rules.

10.3.2.1.6 OCCURS Clause

The OCCURS clause defines a data structure that repeats.

See the topic titled "OCCURS clause" of the "Data Description Entry" section of Chapter 5, the "Data Division" for details.

Format

OCCURS integer-1 TIMES

Syntax Rules

The OCCURS clause can be specified for any screen item.

General Rules

1. When the OCCURS clause is specified, the screen items belonging to that group item are defined repeatedly integer-1 times.
2. When the absolute position is specified in an elementary item belonging to the group item where the OCCURS clause is specified, the same screen position is specified repeatedly for the elementary item.

10.3.2.1.7 OVERLINE Clause

The OVERLINE clause specifies a horizontal line at the top of a screen item.

Format

OVERLINE

Syntax Rules

1. The OVERLINE clause can be specified for any screen item.
2. When the OVERLINE clause is specified at the group item level, it applies to all elementary items belonging to that group item.

General Rules

When the OVERLINE clause is specified for a screen item, a horizontal line is appended at the top of each character screen item.

10.3.2.1.8 PROMPT Clause

The PROMPT clause replaces blank character positions in a screen item by filler characters.

Format

Syntax Rules

1. The PROMPT clause can be specified for input items and update items.
2. When the PROMPT clause is specified at the group item level, it applies to all input and update items belonging to that group item.
3. The filler character, identifier-1, must be one alphabetic, alphanumeric, or national character.
4. The filler character, literal-1, must be an alphanumeric literal, national literal, or figurative constant.

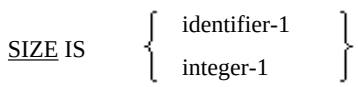
General Rules

1. When the PROMPT clause is specified, blank characters are replaced with filler characters in the input or update item.
2. Characters displayed with the PROMPT clause are not validated as input data.
3. When identifier-1 or literal-1 does not conform to the category specified in the PICTURE clause, results are undefined.
4. If identifier-1 and literal-1 are omitted, the PROMPT clause is ignored.
5. When a character is entered that is the same as the character specified by identifier-1 or literal-1, the entered character is replaced with a blank and stored.

10.3.2.1.9 SIZE Clause

The SIZE clause specifies the screen item logical size.

Format



Syntax Rules

1. The SIZE clause can only be specified for elementary items.

2. The SIZE clause cannot be specified for a numeric data item or a numeric edited data item.
3. identifier-1 must describe an integer.

General Rules

1. When the value specified in the SIZE clause is 0 or a negative integer, the SIZE clause is ignored.
2. When the value specified in the SIZE clause is less than the value implicitly indicated by the related PICTURE clause or VALUE clause, only that screen item part from the left end of the specified size is validated.
3. When the value specified in the SIZE clause is greater than the value implicitly indicated by the related PICTURE clause or VALUE clause, spaces are displayed to the right of the screen item.

10.3.2.1.10 ZERO-FILL Clause

The ZERO-FILL clause specifies that characters not input are to be replaced with zeros instead of spaces.

Format

ZERO-FILL

Syntax Rules

1. The ZERO-FILL clause can be specified for input or update items.
2. When the ZERO-FILL clause is specified at a group item level, it applies to all input and update items belonging to that group item.

General Rules

When the ZERO-FILL clause is specified, if the data item size received from the screen is less than the specified screen item size, all excessive spaces to the left are filled with zeros. If the JUSTIFIED clause is concurrently specified, all excessive spaces to the left are filled with zeros.

10.3.3 Procedure Division

This section describes the following statements:

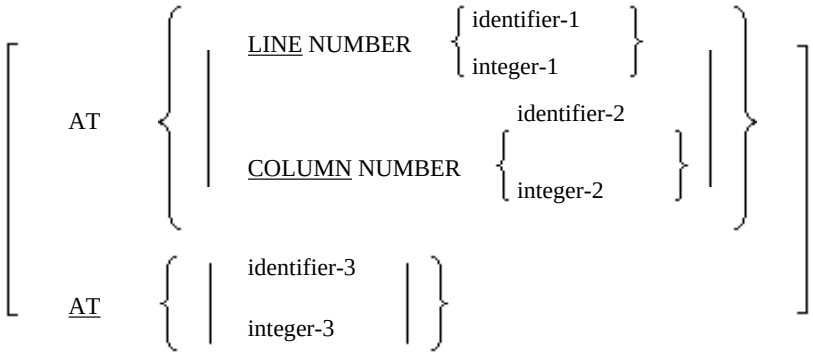
- ACCEPT statement (for manipulating screens)
- DISPLAY statement (for manipulating screens)

10.3.3.1 ACCEPT Statement (Screen Operation)

The ACCEPT statement enters data from the screen.

Format 1

ACCEPT data-name-1



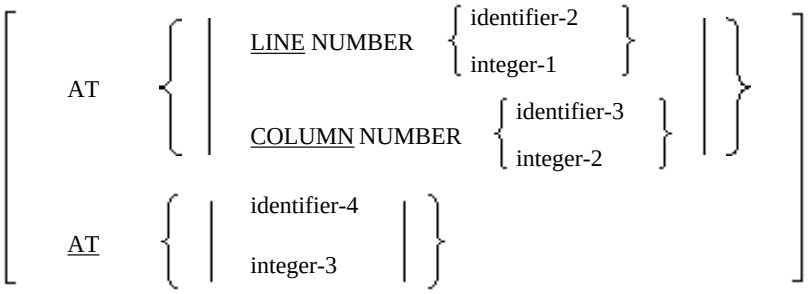
[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-ACCEPT]

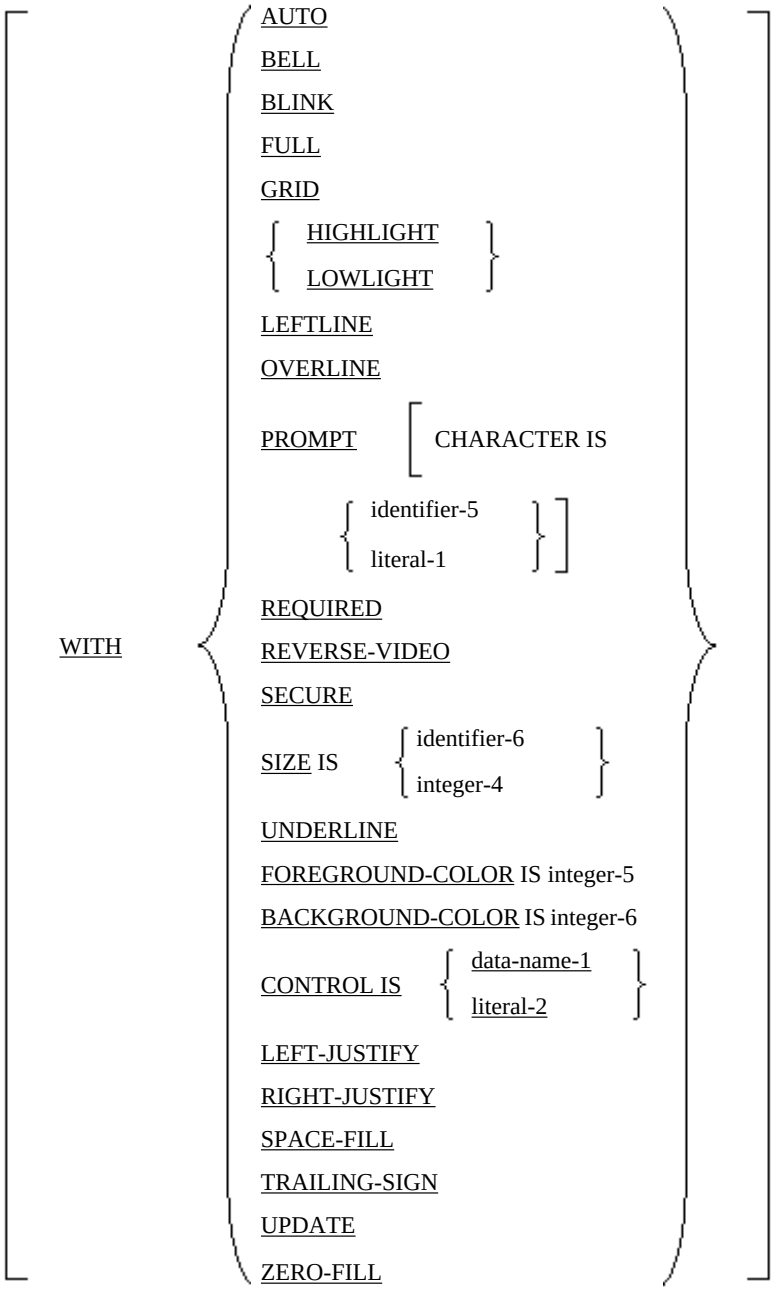
Format 2

ACCEPT identifier-1-1



[FROM CRT]

[MODE IS BLOCK]



[ON EXCEPTION imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-ACCEPT]

Syntax Rules

- Rules for Format 1

  1. identifier-3 must be an unsigned zoned decimal data item of four or six digits.
  2. integer-3 must be a data item of four or six digits.
  3. See the topic titled "ACCEPT Statement (Screen Operation)" of the "Statements" section of Chapter 6, the "Procedure Division" for other syntax rules.

- Rules for Format 2

  1. identifier-1 must be an item defined in the following:

     - WORKING-STORAGE SECTION

  2. identifier-1 must be a fixed-length group item or elementary item. Alphabetic characters, alphanumeric characters, zoned decimals, or national data items can be declared as elementary items.
  3. Words and clauses following identifier-1 can be specified in any order.
  4. integer-1, integer-2, and integer-3 must be unsigned.
  5. identifier-1, identifier-2 and identifier-3 must be unsigned integer data items.
  6. identifier-4 must be an unsigned zoned decimal data item of four or six digits.
  7. integer-3 must have four or six digits.
  8. The SPACE-FILL, ZERO-FILL, LEFT-JUSTIFY, RIGHT-JUSTIFY, and TRAILING-SIGN clauses can only be specified for elementary items.
  9. data-name-1 must be an alphanumeric data item.
  10. literal-2 must be a nonnumeric literal.

General Rules

- Rules for Format 1

  See the topic titled "ACCEPT Statement (Screen Operation)" of the "Statements" section of Chapter 6, the "Procedure Division" for general rules.

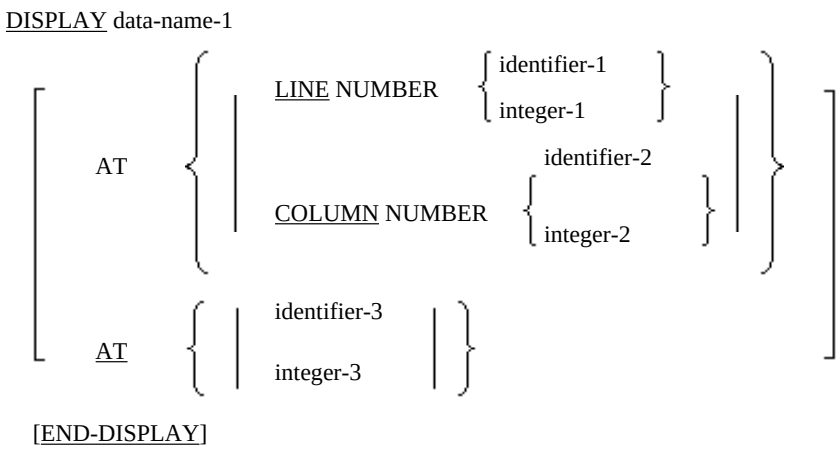
- Rules for Format 2

  1. The ACCEPT statement reads data from the area of the screen corresponding to the item specified in identifier-1.
  2. Press the Enter key to complete execution of the ACCEPT statement.
  3. AT specifies the location of the item to be input on the physical screen.
  4. If the AT is omitted, the item is input from column 1 on line 1.
  5. If identifier-4 is four-digits in length, the line number is specified in the first two digits and the column number in the remaining two digits. If identifier-4 is six-digits in length, the line number is specified in the first three digits and the column number in the remaining three digits.
  6. If integer-3 is four-digits in length, the line number is specified in the first two digits and the column number in the remaining two digits. If integer-3 is six-digits in length, the line number is specified in the first three digits and the column number in the remaining three digits.
  7. The FROM CRT clause specifies that a format 2 ACCEPT statement is being used.
  8. When identifier-1 is a variable-length item, it is always assumed to be maximum length.
  9. When identifier-1 is a group item and the MODE IS BLOCK clause is specified, it is handled as one elementary item.
  10. When identifier-1 is a group item and the MODE IS BLOCK clause is omitted, the elementary item is assumed to belong to that group item. However, fillers and items defined for purposes other than display are ignored.
  11. The LEFT-JUSTIFY, RIGHT-JUSTIFY, SPACE-FILL, TRAILING-SIGN, and UPDATE clauses are treated as comments.
  12. See the section titled "Screen Data Description Entry" in Chapter 5, "Data Division" or the section titled "Screen Data Description Entry" earlier in this chapter for other applicable clauses.

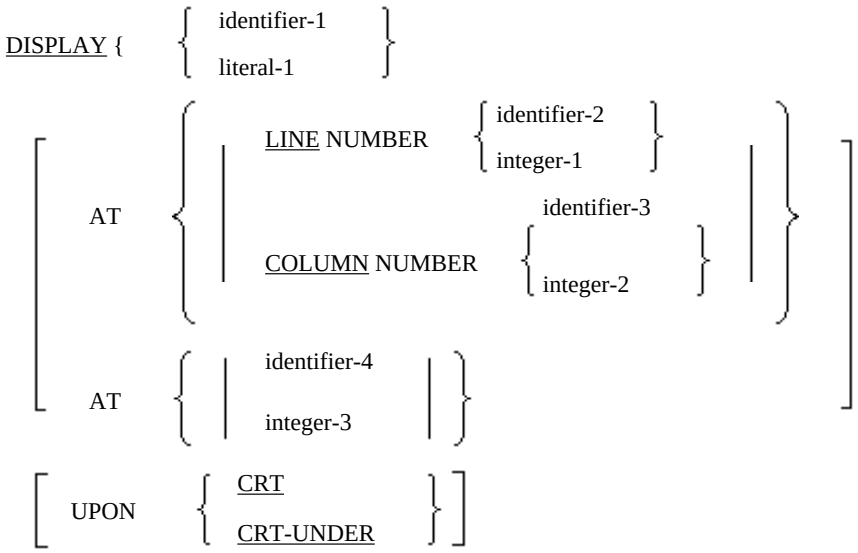
10.3.3.2 DISPLAY Statement (Screen Operation)

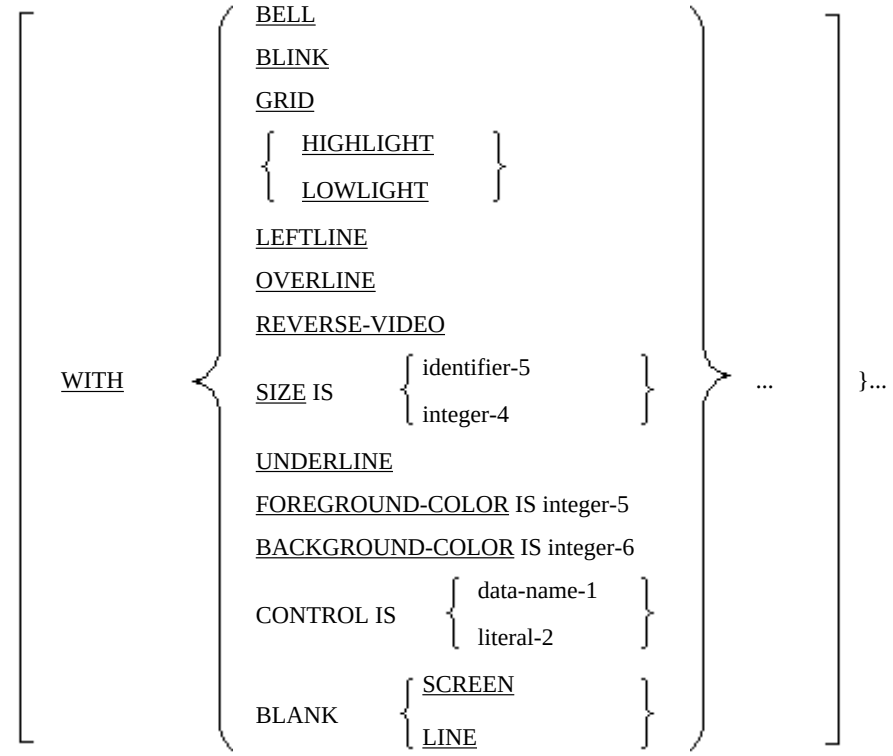
The DISPLAY statement displays data on the screen.

Format 1



Format 2





[END-DISPLAY]

Syntax Rules

- Rules for Format 1

  1. Identifier-3 must be an unsigned zoned decimal item containing four or six digits.
  2. Integer-3 must have four or six digits.
  3. See the topic titled "DISPLAY Statement (Screen Operation)" of the "Statements" section of Chapter 6, the "Procedure Division" for other syntax rules.

- Rules for Format 2

  1. Identifier-1 must be an item defined in the following:

     - WORKING-STORAGE SECTION

  2. Identifier-1 must be a fixed-length group item or elementary item. Alphabetic characters, alphanumeric characters, zoned decimal, or national data item can be declared as elementary items.
  3. Words and clauses following identifier-1 can be specified in any order.
  4. Integer-1, integer-2, and integer-3 must be unsigned.
  5. Identifier-1, identifier-2 and identifier-3 must be unsigned integer data items.
  6. Identifier-4 must be an unsigned zoned decimal data item containing four or six digits.
  7. Integer-3 must have four or six digits.
  8. Data-name-1 must be an alphanumeric data item.
  9. Literal-2 must be a nonnumeric literal.

General Rules

- Rules for Format 1

  See the topic titled "DISPLAY Statement (Screen Operation)" of the "Statements" section of Chapter 6, the "Procedure Division" for general rules.

- Rules for Format 2

  1. The DISPLAY statement displays data on the screen in the location specified in identifier-1.
  2. The AT clause specifies the physical screen position of the item to be input.
  3. When AT is not specified, the item is displayed from column 1 on line 1.
  4. If identifier-4 is four-digits in length, the line number is specified in the first two digits and the column number in the remaining two digits. If identifier-4 is six-digits in length, the line number is specified in the first three digits and the column number in the remaining three digits.
  5. If integer-3 is four-digits in length, the line number is specified in the first two digits and the column number in the remaining two digits. If integer-3 is six-digits in length, the line number is specified in the first three digits and the column number in the remaining three digits.
  6. The UPON CRT clause specifies that a format 2 DISPLAY statement is being used.
  7. The UPON CRT-UNDER clause indicates that displayed items are underlined.
  8. If identifier-1 is a variable-length item, the maximum length of the item is always assumed.
  9. If identifier-1 is a group item and the MODE IS BLOCK clause is specified, it is displayed as one elementary item.
  10. If identifier-1 is a group item and MODE IS BLOCK clause is omitted, the elementary item is assumed to belong to that group item. However, fillers and items defined for purposes other than display are ignored.
  11. See the section titled "Screen Data Description Entry" in Chapter 5, the "Data Division" or the topic titled "Screen Data Description Entries" of the "CONSOLE IS CRT Clause" section earlier in this chapter for other clauses.

10.4 National Functions

This section describes the functions related to the Japanese language offered by the Micro Focus native function.

10.4.1 Procedure Division

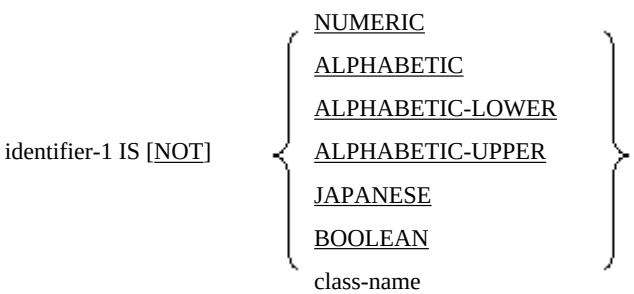
This section describes the class conditions.

10.4.1.1 Class Condition

The class condition is used to check the class of the data item contents.

A JAPANESE check can be specified.

Format



1. The JAPANESE check is true when the data item specified in identifier-1 contains national characters or half-width kana characters.
2. identifier-1 must be one of the following:
   - Data items for display
   - Alphanumeric and national function-identifiers
3. When specifying a JAPANESE check, the following data items cannot be specified in identifier-1.
   - Items specified as numeric characters or Boolean expressions
4. A JAPANESE check cannot be used when the operation mode is Unicode (ISO/IEC 10646-1). Also, a JAPANESE check cannot be done in [.NET].

10.5 Input-Output Statement

This section describes the input-output statements offered by the Micro Focus native function.

10.5.1 Procedure Division

This section describes the following statements:

- READ statement (relative files/indexed files)
- START statement (relative files)
- START statement (indexed files)

10.5.1.1 READ Statement (Relative File and Indexed File)

The READ statement reads records from a file.

Format 1

Reads records sequentially (from relative file and indexed

READ file-name-1 [{ NEXT / PREVIOUS } RECORD]

[INTO identifier-1]

[WITH [NO] LOCK]

[AT END imperative-statement-1]

[NOT AT END imperative-statement-2]

[END-READ]

Format 2

Reads records at any position (from relative file).

READ file-name-1 RECORD

[INTO identifier-1]

[WITH [NO] LOCK]

[INVALID KEY imperative-statement-3]

[NOT INVALID KEY imperative-statement-4]

[END-READ]

Format 3

Reads records at any position (from indexed file).

```
READ file-name-1 RECORD
        [INTO identifier-1]
        [WITH [NO] LOCK]
        [KEY IS {data-name-1} ...]
        [INVALID KEY imperative-statement-3]
        [NOT INVALID KEY imperative-statement-4]
        [END-READ]
```

Syntax Rules

- Rules Common All File Types

  See the topic titled "READ Statement (Sequential file, Relative file, and Indexed file)" of the "Statements" section of Chapter 6, the "Procedure Division" for rules on the READ statement.

- Rules for Relative Files

  1. Format 1 is used to access a sequential file. The PREVIOUS clause cannot be specified for a sequential file.
  2. Format 1 is used to access records in a dynamic access file, using the NEXT clause to read records sequentially.
  3. Format 1 is used to access records in a dynamic access file, using the PREVIOUS clause to read records in reverse sequence.
  4. Format 2 is used to access records in a random access file and for random access to records in a dynamic access file.

- Rules for Indexed Files

  1. Format 1 is used to access a sequential access file. The PREVIOUS clause cannot be specified for sequential access.
  2. Format 1 is used to access records in a dynamic access file, using the NEXT clause to read records sequentially.
  3. Format 1 is used to access records in a dynamic access file, using the PREVIOUS clause to read records in reverse sequence.
  4. Format 3 is used to access records in a random access file and for random access to records in a dynamic access file.

General Rules

- Rules Common to All Formats

  See the Topic titled "READ Statement (Sequential File, Relative File, and Indexed File)" of the "Statements" section of Chapter 6, the "Procedure Division" for information on the rules of the READ statement.

- Rules for Format 1

  1. If a relative or an indexed file in sequential access mode is specified for file-name-1, the NEXT phrase can be omitted, it is assumed.
  2. When a sequential file, sequential access relative file, or sequential access indexed file is specified, the next record is accessed in the file specified for file-name-1.
  3. When a dynamic access relative file or dynamic access indexed file is specified in file-name-1, NEXT specifies the next record in the specified file be read, and PREVIOUS specifies the preceding record in the specified file be read.
  4. When a READ statement is executed, the file position identifier value is determined according to the usable record defining rules. The operation to be performed next is determined by the file position indicator value.

- Rules for Determining Usable Records when PREVIOUS is Omitted

  See the topic titled "READ Statement (Sequential file, Relative file, and Indexed file)" of the "Statements" section of Chapter 6, the "Procedure Division" for more information.

- Rules for Determining Usable Records when PREVIOUS is Specified For Relative Files

  1. The file position indicator value is used by the READ statement to determine usable records according to the following rules. The relative key number is used for record relation in a relative file.

     a. When the file position indicator indicates there is no effective preceding record, the READ statement is not executed successfully.

     b. When the file position indicator indicates there is no optional file, an at end condition occurs.

     c. When the file position indicator was set by a previously executed OPEN statement, an at end condition occurs.

     d. When the file position indicator was set by a previously executed START statement, the record with a relative record number equal to the file position indicator is selected.

     e. When the file position indicator was set by a previously executed READ statement, the first record with a relative record number less than the file position indicator is selected.

  2. If a record satisfying conditions (1) (d) or (e) is not found, an at end condition occurs. A value indicating there is no next record is moved to the file position indicator.

  3. When a record satisfying conditions (1) (d) or (e) (called a selected record) is found, the following processing is performed:

     a. When a RELATIVE KEY phrase is declared for file-name-1 and the number of significant digits in the relative record number of the selected record is greater than the relative key data item size, a value indicating this condition is moved to the file position indicator and an at end condition occurs.
     When the number of significant digits in the relative record number is greater than the relative key item size, the selected record is made usable in the record area specified in file-name-1. In this case, the relative record number of the usable record is moved to the file position indicator. The relative record number of the usable record is transcribed to the relative key item according to transcription rules.

     b. When a RELATIVE KEY phrase is declared in file-name-1, the selected record is made usable in the record specified in file-name-1. In this case, the relative record number of the usable record is moved to the file position indicator.

- Rules for Determining a Usable Record when PREVIOUS is Specified For an Indexed File

  1. The file position indicator value is used by a READ statement for determining usable records. Records in an indexed file are compared using the current reference key values collating sequence.

     a. When the file position indicator indicates there is no preceding record, the READ statement is not executed successfully.

     b. When the file position indicator indicates there is no optional file, an AT END condition occurs.

     c. When the file position indicator was updated by a previously executed OPEN statement, an at end condition occurs.

     d. When the file position indicator was updated by a previously-executed START statement, a record satisfying one of the following conditions is selected:

        - A record positioned by the START statement

        - When a record satisfying this condition is not found or is not the selection object, the record logically following or preceding that record is selected. Whether the record following or preceding that record is selected depends on the relation condition specified in the START statement. However, if the file position indicator is updated by a START statement with the REVERSED ORDER phrase, the record logically preceding that record is selected. This selected record is logically the next record in the direction of the end record to the first record in the file.

e. When the file position indicator was updated by a previously-executed READ statement, a record satisfying one of the following conditions is selected:

   - When searching in the reverse direction by executing a START statement with the REVERSED ORDER clause, or when the PREVIOUS clause has been specified, the record logically preceding the record made usable by the previously executed READ statement is selected. This record is logically the next record in the direction of the end record to the first record in the file.

   - When searching in the reverse direction by executing a START statement with the REVERSED ORDER clause and when the PREVIOUS clause has also been specified, the record logically following the record made usable by the previously executed READ statement is selected.

2. If a record satisfying conditions (1) (d) or (e) is not found, an at end condition occurs. A value indicating there is no next record is moved to the file position indicator.

3. When a record satisfying condition (1) (d) or (e) is found, the record is made usable in the record area specified in file-name-1 and the current reference key value of the record made usable is moved to the file position indicator.

4. When a READ statement is not executed successfully, the reference key value is not updated.

5. When a READ statement is executed using the prime record key or alternate record key as the reference key, the record holding the reference key value may be duplicated. In this case, records are accessed in the following order:

   a. When the file position indicator has been updated by a previously executed START statement with the REVERSED ORDER clause, or when the PREVIOUS clause has been specified in a READ statement with the file positioned on the last record when duplicated records created by the relation conditions specified in the START statement exist, records are accessed in reverse order to the order used when records having duplicated reference key values are created using the WRITE or REWRITE statement.

   b. In other cases, records are accessed in the order in which records with duplicated reference key values are created using the WRITE or REWRITE statement.

6. When a format 1 READ statement is executed after execution of a format 3 READ statement for a dynamic access file, the reference key set by execution of the format 3 READ statement is used in the format 1 READ statement until the reference key is changed, and records are accessed in the following search direction:

   a. Records are accessed in normal order until a START statement with a REVERSED ORDER phrase is executed.

   b. When a START statement with a REVERSED ORDER phrase is executed, records are accessed in reverse order.

   c. When the PREVIOUS phrase is specified, the record logically preceding the current record is accessed. However, if the REVERSED ORDER phrase has been specified with the preceding START statement, records are logically searched in reverse order, records are accessed in normal order.

- Transfer of Control for an Exception Condition

1. If an AT END condition occurs during READ statement execution, the READ statement is not executed successfully. After a value indicating the AT END condition is moved to the I-O status of file-name-1, control is transferred according to the rules described in the topic titled "AT END Phrase" in the "Common Statement Rules" section of Chapter 6, the "Procedure Division".

2. During READ statement execution, if the file position indicator indicates the next record is not found, when the number of significant digits in the relative record number is greater than the size of the relative key item, or when no optional file exists, the following operations are performed in the order specified. However, if the PREVIOUS phrase is specified, the following operations are performed in order when the file position indicator indicates no preceding record is found or there is no optional file:

   a. The file position indicator and I-O status of file-name-1 are updated.

   b. Control is transferred according to the rules described in the topic titled "AT END Phrase" in the "Common Statement Rules" section of Chapter 6, the "Procedure Division".

3. If neither an at end condition or any other exception condition occurs during execution of a READ statement, the following processing is performed in the order specified:

   a. The file position indicator and I-O status of file-name-1 are updated.

   b. Records read are made usable in the record area. When the INTO phrase is specified, they can be implicitly transcribed.

   c. Control is transferred according to rules described in the topic titled "AT END Phrase" of the "Common Statement Rules" section of Chapter 6, the "Procedure Division".

- Rules for Format 2 and Format 3

  See the topic titled "READ Statement (Sequential file, Relative file, and Indexed file)" in the "Statements" section of Chapter 6, the "Procedure Division".

10.5.1.2 START Statement (Relative File)

The START statement logically positions a file for sequential access to records.

Format

START file-name-1

[KEY { IS EQUAL TO / IS = / IS GREATER THAN / IS > / IS NOT LESS THAN / IS NOT < / IS GREATER THAN OR EQUAL TO / IS >= / IS LESS THAN / IS < / IS NOT GREATER THAN / IS NOT > / IS LESS THAN OR EQUAL TO / IS <= } data-name-1]

[INVALID KEY imperative-statement-1]

[NOT INVALID KEY imperative-statement-2]

[END-START]

Note

"=", ">", "<", ">=", and "<=" are key words. These symbols are not underlined to avoid confusion with other symbols.

See the topic titled "START Statement (Relative file)" of the "Statement" section of Chapter 6, the "Procedure Division" for syntax and general rules.

10.5.1.3 START Statement (Indexed File)

The START statement logically positions a file for sequential access to records.

Format 1

Logically positions a file for sequential access of records in normal order.

START file-name-1

```
                    IS EQUAL TO
[                   IS =                                  ]
[                   IS GREATER THAN                       ]
[                   IS >                                  ]
[                   IS NOT LESS THAN                      ]
[                   IS NOT <                              ]
[      KEY          IS GREATER THAN OR EQUAL TO           ] {data-name-1} ...
[                   IS >=                                 ]
[                   IS LESS THAN                          ]
[                   IS <                                  ]
[                   IS NOT GREATER THAN                   ]
[                   IS NOT >                              ]
[                   IS LESS THAN OR EQUAL TO              ]
                    IS <=
```

[INVALID KEY imperative-statement-1]

[NOT INVALID KEY imperative-statement-2]

[END-START]

Format 2

Logically positions a file for sequential access to records in reverse order.

START file-name-1

```
                    IS EQUAL TO
[                   IS =                                  ]
[                   IS LESS THAN                          ]
[                   IS <                                  ]
[      KEY          IS NOT GREATER THAN                   ] {data-name-1} ...
[                   IS NOT >                              ]
[                   IS LESS THAN OR EQUAL TO              ]
[                   IS <=                                 ]
```

WITH REVERSED ORDER

[INVALID KEY imperative-statement-1]

[NOT INVALID KEY imperative-statement-2]

[END-START]

Format 3

Logically positions a file on the record at the start or the end of the reference key.

START file-name-1

FIRST RECORD

[KEY IS {data-name-1}...]

[WITH REVERSED ORDER]

[INVALID KEY imperative-statement-1]

[NOT INVALID KEY imperative-statement-2]

[END-START]

Note

"=", ">", "<", ">=", and "<=" are key words. These symbols are not underlined to avoid confusion with other symbols.

Syntax Rules

See the topic titled "START Statement (Indexed file)" of the "Statements" section of Chapter 6, the "Procedure Division".

General Rules

- Rules Common to All Formats

  See the topic titled "START Statement (Indexed file)" of the "Statements" section of Chapter 6, the "Procedure Division".

- Rules Common to Formats 1 and 2

  1. The record key contained in the file of file-name-1 is compared with the following data items using the conditions specified in the KEY phrase.

     a. When the KEY phrase is specified, the record key is compared with the data contained in data-name-1.

     b. When the KEY phrase is omitted, the record key is compared with the data item specified in the RECORD KEY clause of file-name-1. "IS EQUAL TO" is assumed as the relational operator for the KEY phrase.

  2. In the comparison described in (1), data items are compared with the record reference keys in the file arranged in ascending order according to the character collating sequence. When operands have different lengths, the right side of the longer operand is truncated to be equal to the length of the shorter operand for mnemonic comparison. As a result of the comparison, the following processing is performed:

     a. In format 1, the value indicated by the first or last record satisfying the relation condition is moved to the file position indicator. In this case, whether the value of the first or last record satisfying the relation condition depends on the relational operator used.

     b. In format 2, the value indicated by the last record satisfying the relation condition is moved to the file position indicator.

     c. If no records in the file satisfy these conditions, an invalid key condition occurs and the START statement is not executed successfully.

- Rules for Format 3

  See the topic titled "START Statement (Indexed File)" of the "Statements" section of Chapter 6, the "Procedure Division".

10.6 Inter-program Communication Function

10.6.1 PROCEDURE DIVISION

The following statement is described here.

- CALL statement (inter-program communication)

10.6.1.1 CALL Statement (Inter-program Communication)

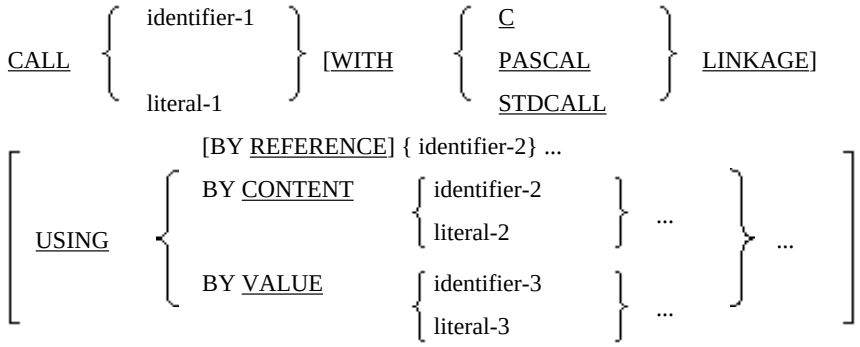
The CALL statement transfers control to another program in the run unit.

Note

This is a function specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF] and [Linux64].

Format 1

ON OVERFLOW phrase



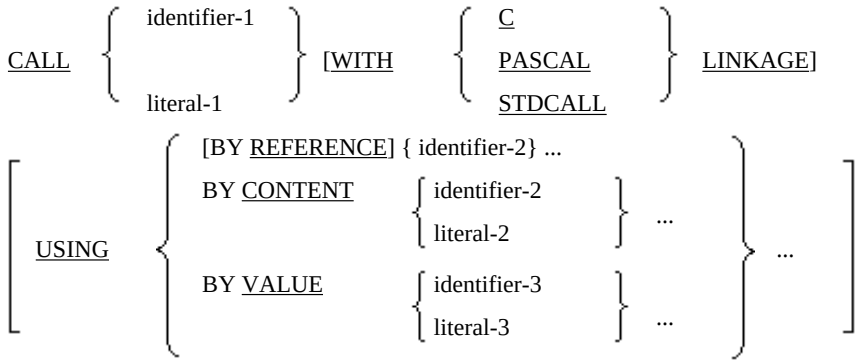
[RETURNING identifier-4]

[ON OVERFLOW imperative-statement-1]

[END-CALL]

Format 2

ON EXCEPTION phrase



[RETURNING identifier-4]

[ON OVERFLOW imperative-statement-1]

[NOT ON EXCEPTION imperative-statement-2]

[END-CALL]

Note

The WITH phrase is specific to Win16 and [Win32].

literal-3 is specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF] and [Linux64].

Syntax Rules

1. literal-3 must be a nine-digit-or-less numeric literal except in [Winx64], [LinuxIPF] and [Linux64], where literal-3 must be an eighteen-digit-or-less numeric literal.
2. See the topic titled "CALL Statement (Inter-program Communication)" of the "Statements" section of Chapter 6, the "Procedure Division" for other syntax rules.

General Rules

See the topic titled "CALL Statement (Inter-program Communication)" of the "Statements" section of Chapter 6 for general rules.

F

Chapter 11 Object-Oriented Programming Functions

This chapter gives you an overview of the object-oriented programming function and the related coding rules.

Note

The object-oriented programming function is specific to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF] and [Linux64].

11.1 Overview

The object-oriented programming function provides the abilities to create object-oriented programs in COBOL. The function provides you with the capabilities to:

- Define classes including factory and object definitions
- Define factory objects and data encapsulated in objects
- Define methods for factory objects or object instances
- Inherit from existing classes to define new ones
- Polymorphism
- Define data items to store references to objects
- Invoke methods of objects
- Create requested objects
- Use object-oriented methods when you develop new COBOL programs or maintain existing COBOL programs

11.2 Terminology

In this document, the following terms are used to describe the object-oriented programming function:

binding

The process of linking a method invocation to a method implementation. Binding can be done at compile time if the compiler is able to determine the class of the object; in this case it is called static or early binding. Binding that is not done at compile time is called dynamic or late binding and is a form of method resolution that associates a method with an operation at run time, depending on the class of the receiving object.

child class

A class inherits from another class. If two classes have an inheritance relationship between them, the class that inherits is a child class, while the other class from which the child class inherits is the inherited class. Synonymous with a subclass.

class

The entity that defines common behavior and implementation to be shared by zero or more objects. Objects that share the same implementation are assumed to belong to the same class.

class definition

A COBOL source unit that defines a class.

class-name

A user-defined word to identify a class.

compilation group

A sequence of COBOL source units to be compiled together.

conformance

The property that allows an object with a given interface to be used where an object with a different interface is expected. Conformance ensures that any operation specified for the conformed interface is supported by the conforming interface.

exception condition

A condition detected during execution of a program, which indicates that an error or exception to normal operations has occurred.

exception object

An object that acts as an exception condition.

factory definition

A COBOL source unit that defines a factory object.

factory (object)

A single object defined by the factory definition of a class. Each class has one factory object. Factory objects create objects of the class.

factory (object) data

The data for a factory object. This is declared in the DATA DIVISION of a factory definition.

factory (object) identifier

An object identifier to identify a factory object.

factory method

A method for a factory object.

inheritance

A mechanism for using the implementation of one or more classes as the basis for another class. A subclass inherits from one or more super-classes. An inheriting class conforms to the inherited class.

interface (for a method)

Information required to invoke a method properly, including names and parameter specifications. This information can be specified as a method prototype.

invocation operator

A pair of adjacent colon characters, i.e.: '::', used in an in-line invocation of a method.

method

Procedural code declared in the PROCEDURE DIVISION of a method. It is executed by a method invocation on that object.

method data

The data declared in the DATA DIVISION of a method definition.

method definition

A COBOL source unit that defines a method.

method invocation, invocation

A request for execution of a named method on a given object. A method invocation identifies an object, method-name, and the parameters required by a method definition.

method-name

A user-defined word to identify a method.

method prototype

The method-names and parameter types (including a returning item if specified) for a method.

method prototype definition

A COBOL source unit that defines a method prototype.

object data

The data described in the DATA DIVISION for an object definition. The data described in a method are excluded.

object definition

A COBOL source unit that defines an object.

object identifier

An identifier to identify an object. This is also referred to as an object reference identifier.

object (instance)

A unit consisting of data, and methods that act on that data.

object method

A method of an object (as opposed to a factory method).

object property

A name qualified by an object reference, which is used to get a value from or pass a value to an object.

object reference identifier

An identifier to identify an object, by referencing the object.

parent class

A class from which another class inherits. When two classes have an inheritance relationship between them, the class that inherits is a child class while the other class from which that child class inherits is a parent class. Synonymous with a super-class.

predefined object identifier

The reserved words EXCEPTION-OBJECT, NULL, SELF, and SUPER used to identify particular objects.

property-name

A property-name identifies an object property, which is a means of passing/getting information to/from an object.

returning item

A data item used to return the execution result of a program or method.

source element

A sequence of statements excluding the other source units included in a source unit.

source unit

A sequence of statements from the IDENTIFICATION DIVISION through to the end program marker or the end of a compilation group, including any contained source units.

special class

A class to which a special object belongs.

special class of interface specification

A class to which a special object having a special interface belongs.

special object

An object that is not a COBOL object, such as an OLE object.

subclass

A class that inherits from another class. If two classes have an inheritance relationship between them, the class that inherits is a subclass while the other class from which the subclass inherits is a super-class. Synonymous with a child class.

Super-class

A class from which another class inherits. When two classes have an inheritance relationship between them, the class that inherits is a subclass while the other class from which the subclass inherits is a super-class. Synonymous with an inherited class.

11.3 General Rules

This section describes the concepts of the COBOL language native to the object-oriented programming function and its general rules. Refer to the corresponding part of Chapter 1, "General Rules" when you use the object-oriented programming function.

11.3.1 Basic Elements of the Language

This section describes COBOL words and figurative constants native to the object-oriented programming function.

11.3.1.1 COBOL Words

These COBOL words are described below:

- User-defined words
- Reserved words

11.3.1.1.1 User-Defined Words

There are three types of user-defined words native to the object-oriented programming function.

- Class-name
- Method-name
- Property-name

Class-name

A class-name identifies a class that defines common behaviors and attributes to be shared by zero or more objects.

Method-name

A method-name identifies a method.

Property-name

A property-name identifies an object property that is a means of getting information out of and passing information back into an object.

11.3.1.1.2 Reserved Words

This section describes the reserved words native to the object-oriented programming function. Refer to Appendix A "Reserved Word List" for a complete listing of reserved words. An asterisk (*) in the Remarks column indicates that the reserved word is one added for this object-oriented programming function only.

Special Purpose Words

The special purpose word is:

- Predefined object identifier

Predefined object identifier

Some reserved words are used as predefined object identifiers.

The predefined object identifiers are:

- EXCEPTION-OBJECT
- NULL
- SELF
- SUPER

11.3.1.2 Figurative Constants

1. Except for program definitions, the figurative constant ALL literal, when the length of the literal is greater than one, should not be associated with numeric or numeric-edited data item.
2. A figurative constant cannot be specified where an alphanumeric literal is used to identify a method.

11.3.2 Concept of Data Description Entry

This section describes the concept of data description entries native to the object-oriented programming function.

11.3.2.1 Concept of Class

All the elementary data items, literals, and functions have classes and categories. Table 11.1 "Relationship between Class and Category for Object Data Items" shows the relationship between class and category for object data items.

Table 11.1 Relationship between Class and Category for Object Data Items

Category	Class
Object-reference	Object

Note

In general, the characters in a group item are alphanumeric. However, when a group item contains an object reference data item, the group item cannot contain characters.

11.3.3 Uniqueness of Reference

This section describes uniqueness of reference native to the object-oriented programming function.

11.3.3.1 Identifier

To reference a data-name uniquely, the data-name should be given in the following formats:

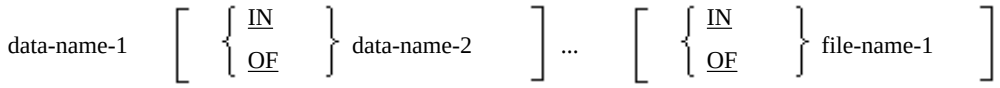
Format 1

(Function-identifier)

function-identifier-1

Format 2

(Qualified-data-name-with-subscripts)



[{subscript-1}...]

Format 3

(Reference modifier)

identifier-1 reference-modifier-1

Format 4

(In-line method invocation)

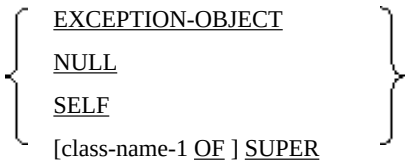
in-line invocation-1

Format 5

(Object-modifier)

identifier-2 object-modifier-1

Format 6 (Predefined object identifier)



Format 7

(Object property)

property-name-1 OF identifier-3

Format 8

(Class-name)

class-name-1

Format 9

(Pointer qualifier)

Pointer-qualifier data-name-1

Syntax Rules

- All Formats

  Identifiers are defined recursively. Identifier-1, identifier-2, and identifier-3 can take any format for identifiers.

- Format 1

  Function-identifier-1 is a function-identifier defined in topic "Function Identifier" in the "Uniqueness of Reference" section of Chapter 1, "General Rules".

- Format 2

  1. The words IN and OF are equivalent.
  2. If data-name-1 is not unique in the scope of names of the current source element, it should be followed by a syntactically correct combination of qualifiers and subscripts required for uniqueness of reference. See topic "Scope of Names" in the "Inter-program Communication Module" section of Chapter 2, "COBOL Modules" and the "Scope of Names" topic below.
  3. Refer to the topic "Subscripting" of the "Concept of Data Description" in Chapter 1, "General Rules" for the definition of subscripts.

- Format 3

  Reference-modifier-1 is described in the topic "Reference Modification" in the "Uniqueness of Reference" section of Chapter 1, "General Rules".

- Format 4

  In-line-invocation-1 is described in topic "In-line Method Invocation" below.

- Format 5

  See the topic "Object-Modifier" below for the details of object-modifier.

- Format 6

  See the "Predefined Object Identifier" topic below for the details of predefined object identifiers.

- Format 7

  See the topic "Object Property" below for the details of object property.

- Format 8

  Class-name-1 is described in the topic "Class name" below.

- Format 9

  See the topic "Pointer" of the "Uniqueness of Reference" section of Chapter 1, "General Rules" for the details of pointer qualifiers.

General Rules

1. The following rules for uniqueness of reference apply in the order listed below:
   a. For data-name qualification, IN or OF qualifies the (possibly already qualified) data-name on the left with a file-name, report-name, or data-name on the right.
   b. Subscript applies to the fully qualified data-name on the left.

c. The object-modifier applies to the object identifier on the left.

d. For the object property, OF applies the property-name on the left to the object identifier on the right.

e. The invocation operator for in-line method invocation applies a literal method-name with optional parameters enclosed in parentheses on the right to the object identifier on the left.

f. Without arguments, a function identifier is assumed to be an elementary identifier. If specified with arguments, the function-name will be applied to argument list enclosed in parentheses.

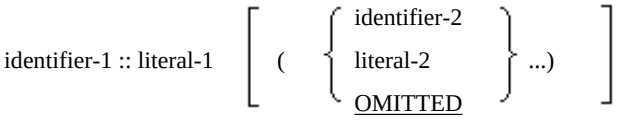
g. The reference modifier applies to the identifier on the left.

2. See the topic "Pointer" of the "Uniqueness of Reference" section of Chapter 1, "General Rules " for the order in which the rules for uniqueness of reference including pointing are applied.

11.3.3.2 In-Line Method Invocation

In-line invocation of a method references the value returned from the invoked method.

Format



Syntax Rules

1. This format can be used anywhere an identifier can be specified except for receiving items. Using this format, and using a data item defined in the same way as the returning parameter of the method specified by the identifier-1 and literal-1, result in the same behavior.
2. Identifier-1 must be an object identifier. However, it must not be a class name or object identifier of a late-bind special class, pointed identifier, or object reference identifier in which the USAGE OBJECT REFERENCE clause with no other optional phrase is specified.
3. Literal-1 must be a nonnumeric literal or national language nonnumeric literal of which the value is the method name specified for identifier-1.
4. The method identified by identifier-1 and literal-1 must have a return item.
5. Identier-2 must be SELF, EXCEPTION-OBJECT, NULL, a class name specified in the REPOSITORY paragraph, or a data item defined in the FILE, WORKING STORAGE, or LINKAGE section. However, it must not be a special class name.
6. Identifier-2 must not be defined by reference modification.
7. Literal-2 must be valid as the sending item of a MOVE statement to the item of the same class and category as the corresponding parameter of the method specified by identifier-1 and literal-1.
8. Identifer-2 must conform to the rules described in topic "Conformance of Parameters" of the "The Procedure Division" section later in this chapter.
9. When an internal Boolean data item is specified at identifier-2, the internal Boolean data item must be defined to place on byte alignment.
10. When OMITTED is specified, identifer-1 must be the class name or object reference identifier of a special class.

General Rules

1. Literal-1 is a name of a method.
2. Identifier-1 is an object on which a method is invoked.

3. Using in-line method invocation and using a temporary data-name after either of the following statements will result in the same behavior:

   INVOKE identifier-1 literal-1 USING arguments
   RETURNING temporary-data-name

   INVOKE identifier-1 literal-1 RETURNING temporary-data-name

   a. If arguments are given, they are parameters specified for in-line method invocation.
   b. A temporary data-name has the same description, class, and category as the returning item for the method specified by identifier-1 and literal-1.
   c. A temporary data-name is a temporary item that exists for in-line invocation only.

4. If an exception occurs during execution of a statement containing this format, the resumption point will be the next executable statement.
5. When ANY LENGTH is specified in the RETURNING argument in the method declaration that is identified by literal-1, the in-line method invocation cannot be used.
6. When OMITTED is specified, the corresponding argument is regarded as omitted.

11.3.3.3 Object-Modifier

An object-modifier is a syntactically correct combination of character-strings and separators that reference an object reference identifier through the specified interface.

Format

identifier-1 AS { [FACTORY OF] class-name-1 [ONLY] | UNIVERSAL }

Note

identifier-1 in the above format is used to denote the context and is not a part of an object-modifier.

Syntax Rules

1. Identifier-1 must be an object reference identifier. However, it should not be a predefined object identifier SUPER, class name or object identifier of special class except for *COM(*OLE) and early bind special class, or pointer.
2. Identifier-1 should not be referenced as a receiving item.
3. If class-name-1 is a special class name, it must be an early bind special class.
4. If identifier-1 is a special class object reference identifier, class-name-1 must be a special class name, and you cannot select options before and after it. You cannot specify UNIVERSAL either.

   If identifier-1 is not a special class object reference identifier, class-name-1 should not be a special class name.

General Rules

1. The object-modifier returns an object that references the object specified by identifier-1 presuming the interface specified by the AS phrase.
2. If class-name -1 is specified with the both optional phrases omitted, the implicit description of the result will be "USAGE OBJECT REFERENCE class-name-1".

   When class-name-1 is not a special class, if the object identified by identifier-1 is neither an object of class-name-1 nor an object of the subclass of class-name-1, a statement containing this code will terminate abnormally when executed.

When class-name-1 is a special class, the rules for that special class depend on whether coclass or dispinterface has been specified.

If coclass has been specified and the object identified by identifier-1 is not the object of that coclass, a statement containing this code will terminate abnormally when executed.

If dispinterface has been specified and the object identified by identifier-1 has not implemented that dispinterface, a statement containing this code will terminate abnormally when executed.

3. If "FACTORY OF class-name-1" is given, the implicit description of the result will be "USAGE OBJECT REFERENCE FACTORY OF class-name-1". If the object identified by identifier-1 is neither a factory object of class-name-1 nor a factory object of the subclass of class-name-1, a statement containing this code will terminate abnormally when executed.
4. If "class-name-1 ONLY" is given, the implicit description of the result will be "USAGE OBJECT REFERENCE class-name-1 ONLY". If the object identified by identifier-1 is not an object of class-name-1, a statement containing this code will terminate abnormally when executed.
5. If "FACTORY OF class-name-1 ONLY" is given, the implicit description of the result will be "USAGE OBJECT REFERENCE FACTORY OF class-name-1 ONLY". If the object identified by identifier-1 is not a factory object of class-name-1, a statement containing this code will terminate abnormally when executed.
6. If "UNIVERSAL" is given, the implicit description of the result will be "USAGE OBJECT REFERENCE", which has no optional phrase indicating the class or interface of the object referenced by identifier-1.

11.3.3.4 Predefined Object Identifier

Predefined object identifiers are:

- EXCEPTION-OBJECT
- NULL
- SELF
- SUPER

11.3.3.4.1 EXCEPTION-OBJECT

EXCEPTION-OBJECT is a predefined object identifier used in declaratives to reference the current exception object.

Format

EXCEPTION-OBJECT

Syntax Rules

1. EXCEPTION-OBJECT can be used only in a declarative procedure associated with a USE statement specifying a class-name.
2. EXCEPTION-OBJECT cannot be used as a receiving operand.

General Rules

1. EXCEPTION-OBJECT references the current exception object. If an exception object is not associated with the current exception, EXCEPTION-OBJECT is the NULL object.
2. EXCEPTION-OBJECT is implicitly declared as "USAGE OBJECT REFERENCE" if EXCEPTION-OBJECT is used at the declare procedure that is associated with the USE statement with a CLASS name other than special CLASS.
3. EXCEPTION-OBJECT is implicitly declared as "Class name related with USAGE OBJECT REFERENCE *OLE-EXCEPTION" if EXCEPTION-OBJECT is used at the declare procedure that is associated with the USE statement with a special CLASS name.

11.3.3.4.2 NULL

NULL is a predefined object identifier to reference a NULL object.

Format

NULL

Syntax Rules

NULL cannot be specified as a receiving item.

General Rules

NULL always references the same object, the NULL object.

11.3.3.4.3 SELF and SUPER

SELF and SUPER are predefined object identifiers to reference the object on which the current method is executing.

Format

```
{ SELF                          }
{ [class-name-1 OF] SUPER       }
```

Syntax Rules

1. This format can be used only in the PROCEDURE DIVISION of the method definition.
2. This format cannot be used as a receiving item.
3. SUPER can be used as an object for in-line method invocation or object property, or as an object for method invocation through the INVOKE statement.
4. class-name-1 must be a class-name specified by the INHERITS clause in the class definition including this code.
5. If two or more class-names are given in the INHERITS clause in the class definition, class-name-1 must be specified.

General Rules

1. Predefined object identifiers, SELF or SUPER, reference the object used to invoke a method in which SELF or SUPER appear.
2. If SELF is specified to invoke a method, the method resolution is based upon the set of methods defined for the object.
3. If SUPER is specified to invoke a method, the method resolution shall ignore all the methods defined in the class containing the invocation and all the methods defined in any subclass of that class.
4. If a class-name is specified, only the method defined for class-name-1 will be searched.

11.3.3.5 Object Property

The object property provides a special syntax to reference the information of an object or to set information on an object.

Format

property-name-1 OF identifier-1

Syntax Rules

1. Identifier-1 must be an object identifier. However, it must not be a class name or object identifier of a late-bind special class, pointed identifier, or object reference identifier in which the USAGE OBJECT REFERENCE clause with no other optional phrase is specified.

2. When an object property is used as a sending item, a get property method with property-name-1 must exist in the object referenced by identifier-1.
3. When an object property is used as a receiving item, a set property method with property-name-1 must exist in the object referenced by identifier-1.
4. An object property to be used as a sending item can be described in the same way as a returning item of the get property method. This object property can be described in any place where a data item can be described with that description as a sending item.
5. An object property to be used as a receiving item can be described in the same way as the USING parameter of the set property method. This object property can be described in any place where a data item can be described with that description as a receiving item.
6. The data description entry for the item specified at the RETURNING phrase of a get property method must be the same as the data description entry for the item specified at the USING phrase of a set property method.
7. An object property must not be defined by reference modification.
8. Object properties, when used as receiving items, can only be specified in:
   - The receiving item of a MOVE statement without a CORRESPONDING phrase
   - The receiving item of a SET statement

General Rules

1. If an object property is only used as a sending item, a conceptual temporary data item, temp-1, will be used there. The value of the property is determined when the associated get property method is executed and a returned value placed in temp-1. The data description for temp-1 is the same as that specified in the RETURNING phrase of a get property method.
2. If an object property is only used as a receiving item, a conceptual temporary data item, temp-2, will be used there. The value of the property is assigned when the associated set property method is executed and the content of temp-2 is moved to the parameter. The data description for temp-2 is the same as that specified at the USING phrase of a set property method.
3. If an exception occurs on a statement with an object property, control is transferred to the next executable statement.

11.3.3.6 Class-Name

A class-name can be also used as an object identifier.

Format

class-name-1

Syntax Rules

1. A class-name cannot be specified as a receiving item.
2. A special class name can only be specified in identier-1 of the INVOKE statement. However, a class name of a special class whose class specifier literal is "*COM-EXCEPTION" (or "*OLE-EXCEPTION") can be specified in class-name-1 of the USE AFTER EXCEPTION statement.

General Rules

Class-name-1 identifies a factory object of class-name-1.

11.3.4 Reference Format

This section describes continuation of lines and the scope for names native to the object-oriented programming function.

11.3.4.1 Continuation of Lines

Two characters for the invocation operator "::" must be on the same line.

11.3.4.2 Scope of Names

This section describes the scopes of class-names and method-names.

Scope of Class-Names

A class-name for a class used in a source element must be declared in the REPOSITORY paragraph of that or a containing source element.

Within a compilation group, there must be one and only class definition for a class-name.

A class-name declared in the REPOSITORY paragraph of a source element can be used in that source element and any nested source units.

Scope of Method-Names

A method-name for a method is declared at the METHOD-ID paragraph. Method-names can be referenced only in the INVOKE statement, in-line invocation, and END METHOD header.

A method declared in a class definition must have a unique method-name within that class definition. A method declared in the inheriting class can have the same name as the methods of the inherited class, in compliance with the requirements described in topic "Method-Id Paragraph" of the "Identification Division and End Marker" section later in this chapter.

11.4 Compilation Group Structure

This section describes the structure of programs native to object-oriented programming, and concepts used by the object-oriented programming function.

11.4.1 Organization

One compilation group can include one or more COBOL source units.

Statements, entries, paragraphs, and sections of a source unit are grouped into the four divisions appearing in the following order, except for source text manipulation statements and end markers:

1. IDENTIFICATION DIVISION
2. ENVIRONMENT DIVISION
3. DATA DIVISION
4. PROCEDURE DIVISION

An appropriate division header indicates the beginning of a division in a source unit. The beginning of the IDENTIFICATION DIVISION can be denoted by one of the paragraph headers allowed for the IDENTIFICATION DIVISION.

The end of a division in a source unit is indicated by:

- The beginning of the succeeding division within that source unit
- The end marker of that source unit
- Its physical position after where no more source lines occur

The end of a COBOL source unit is indicated explicitly by an end marker, or implied by the fact that no more source lines follow.

A source unit directly or indirectly included in another source unit is assumed to be a different source unit that additionally references resources of the source unit including it.

11.4.1.1 COBOL Compilation Group

This section describes a COBOL compilation group.

Format

[{source unit}...]

Source Unit

Format

{ program definition
  class definition
  method definition }

Program Definition

Format

[IDENTIFICATION DIVISION.]
PROGRAM-ID. { program-name-1 [AS literal-1]
              Program-name-literal }
[IS { COMMON
       INITIAL } PROGRAM].
[ENVIRONMENT DIVISION]
[DATA DIVISION]
[PROCEDURE DIVISION
    [{program definition}...]]
[END PROGRAM { program-name-1
               program-name-literal } .]

Class Definition

Format

[IDENTIFICATION DIVISION.]
CLASS-ID. class-name-1 [AS literal-1]
        [INHERITS {class-name-2}...].
[ENVIRONMENT DIVISION]
[factory definition]
[object definition]
END CLASS class-name-1.

Factory Definition

Format

[IDENTIFICATION DIVISION .]
FACTORY.
[ENVIRONMENT DIVISION]

```
[DATA DIVISION]
[ PROCEDURE DIVISION.
  [ [ method definition
    | method prototype definition ] ... ] ]
END FACTORY.
```

Object Definition

Format

```
[ IDENTIFICATION DIVISION .]
OBJECT.
[ENVIRONMENT DIVISION]
[DATA DIVISION]
[ PROCEDURE DIVISION.
  [ [ method definition
    | method prototype definition ] ... ] ]
END OBJECT.
```

Method Definition

Format

```
[ IDENTIFICATION DIVISION. ]
METHOD-ID. { method-name-1 [AS literal-1]
             { GET | SET } PROPERTY property-name-1 }
           [ { | OVERRIDE
               | OF class-name-1
               | OF [ FACTORY OF ] class-name-1 } ] .
[ENVIRONMENT DIVISION]
[DATA DIVISION]
[PROCEDURE DIVISION]
END METHOD [method-name-1].
```

Method Prototype Definition

Format

```
[ IDENTIFICATION DIVISION. ]
METHOD-ID. { method-name-1 [AS literal-1]
             { GET | SET } PROPERTY property-name-1 }
           [ { | OVERRIDE
               | PROTOTYPE } ] .
[DATA DIVISION]
[PROCEDURE DIVISION]
```

END METHOD [method-name-1].

Program Definition Description

A program definition is a source unit beginning with the IDENTIFICATION DIVISION containing a PROGRAM-ID paragraph. A program definition can include the ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. A program definition can end with an END PROGRAM header.

Refer to the "PROGRAM-ID Paragraph" topic of the "Composition of the Identification Division" section of Chapter 3, "Identification Division and End Program Header" for the details of a program definition.

Class Definition Description

A class definition is a source unit beginning with an IDENTIFICATION DIVISION containing a CLASS-ID paragraph. A class definition can include the ENVIRONMENT DIVISION, factory definition, and object definition. Definitions in the ENVIRONMENT DIVISION apply to the class definition and included definitions. The ENVIRONMENT DIVISION must not contain an INPUT-OUTPUT SECTION. A class definition must end with an END CLASS header.

Factory Definition Description

A factory definition is a source unit beginning with an IDENTIFICATION DIVISION containing a FACTORY paragraph. A factory definition can include the ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. Definitions in the ENVIRONMENT DIVISION apply to the factory definition and all the methods declared in the factory definition. The ENVIRONMENT DIVISION must not contain the CONFIGURATION SECTION. The DATA DIVISION can only contain the FILE SECTION, WORKING-STORAGE SECTION, CONSTANT SECTION, and BASED-STORAGE SECTION. The PROCEDURE DIVISION can contain only a method definition and it cannot contain other statements and section headers or paragraph headers. A factory definition must end with an END FACTORY header.

Object Definition Description

An object definition is a source unit beginning with an IDENTIFICATION DIVISION containing an OBJECT paragraph. An object definition can include the ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. Definitions in the ENVIRONMENT DIVISION apply to the object definition and all the method declared in the object definition. The ENVIRONMENT DIVISION must not contain the CONFIGURATION SECTION. The DATA DIVISION can only contain a FILE SECTION, WORKING-STORAGE SECTION, CONSTANT SECTION, and BASED-STORAGE SECTION. The PROCEDURE DIVISION can contain only a method definition and it cannot contain other statements and section headers or paragraph headers. An object definition must end with an END OBJECT header.

Method Definition Description

A method definition is a source unit beginning with an IDENTIFICATION DIVISION containing a METHOD-ID paragraph. A method definition can include the ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. The ENVIRONMENT DIVISION can contain a REPOSITORY paragraph only when a method definition is not included in a class definition. The DATA DIVISION can only contain a FILE SECTION, WORKING-STORAGE SECTION, CONSTANT SECTION, BASED-STORAGE SECTION, and LINKAGE SECTION. The PROCEDURE DIVISION should conform to the rules described in the PROCEDURE DIVISION of a program. A method definition must end with an END METHOD header.

Method Prototype Definition Description

A method prototype definition is a method definition that has the PROTOTYPE attribute. The DATA DIVISION of a method prototype definition can only contain the LINKAGE SECTION. The PROCEDURE DIVISION of a method prototype definition must only contain a PROCEDURE DIVISION header. A method prototype definition must end with an END METHOD header.

11.4.2 External Repository

An external repository can store all the information required by a compiler, including class-names, method-names, and method parameters. The compiler is stored in the external repository when compiling a class-definition. At that time,

information required to compile source elements referencing that class is also included. Refer to the topic "REPOSITORY Paragraph" of the "The Environment Division" section later in this chapter for further details.

11.4.3 Program Organization and Communication

A compilation group is a sequence of syntactically correct COBOL statements. A compilation group contains one or more source units. A source unit can include other source units in it. Those included source units can reference the resources of the source unit including them. Refer to the topic "COBOL Compilation Group" of the "Compilation Group Structure" section later in this chapter for further details.

If Source Unit A is included in Source Unit B, it does not matter whether it is included directly or indirectly. If there is no source unit which is included in Source Unit A and includes Source Unit B, B is directly included in A. If there is a source unit which is included in Source Unit A and, at the same time, includes Source Unit B, B is included in A indirectly.

11.4.3.1 Objects and Classes

An object consists of data and methods. Every object belongs to a class. A class describes data structure and methods to be applied to all objects belonging to that class. A class also has a factory object that has data and methods. A factory object is an object to create objects.

11.4.3.2 Object References

An object reference is a value to identify the object uniquely during the lifetime of the object. Two objects do not have the same object reference value.

11.4.3.3 Methods

The procedural code in an object is placed in a method. Each method has its own method-name, DATA DIVISION, and PROCEDURE DIVISION. If a method is invoked, the procedural code contained in the method is executed. A method is invoked by specifying an identifier to reference an object and a method-name. For methods, you can specify parameters and a returning item.

11.4.3.4 File Connector

This section describes file connectors for a method, object, and factory object.

Files within a Method

A method definition can have internal file description entries. If a method is invoked recursively, each method invocation will have its own file connectors given the internal attribute. If declaratives and open internal files are not closed when the method is complete, these files will be closed by the system, as is the case with an executed CLOSE statement with no optional phrases.

Files within an Object

An object definition can have internal file description entries. If there are two or more objects in the same class, each object will have its own file connectors. If an object is deleted and the files declared within the object are not closed, all the files will be closed by the system.

Files within a Factory Object

A factory object definition can have internal file description entries. If the files defined within the factory object are not closed when the run unit is complete, these files will be closed by the system.

11.4.3.5 External Name and Internal Name

Classes and methods have both an external and internal name.

The external name is used when referencing other classes or methods, and when other programs or classes reference it. Because the external name can be defined and referenced by a literal, the external name becomes the means to use the names that are outside the scope of the COBOL language when linking with other languages or other systems.

The internal name is the name that is defined according to the rule of user-defined words. The internal name is used to express an arbitrary class-name or method-name within a compilation unit.

When an external name is not explicitly defined for a class or method, it is assumed that an external name having the same name as the internal name is implicitly defined. In this document, any name not stated to be an external name is an internal name.

11.4.3.6 Global Names

Data-names and file-names declared in the DATA DIVISION of an object or factory definition are global names.

11.4.3.7 Method Invocation

The procedural code of a method is executed when the method is invoked. If a method is invoked on an object, the invocation will be bound with a method implementation, based on the first applicable rule in the following list:

1. If the method with the method-name specified in the invocation has been defined in the object definition for an object, that method will be bound.
2. If the method with the method-name specified in the invocation has been defined in one of the classes which is inherited by the class of the object, that method will be bound. A class is searched for those classes in the order specified by the INHERITS phrase.
3. If no method is found, the statement including this code will terminate abnormally when executed.

If a method is defined in the class or another class from which that class inherits directly or indirectly, it is assumed that method is defined in that class.

11.4.3.8 Method Results

If a method definition has the RETURNING phrase in its PROCEDURE DIVISION header, the contents of the data item in the RETURNING phrase when the method terminates becomes the result of the method. When a method is invoked by the INVOKE statement, the result is placed in the identifier given in the RETURNING phrase of the INVOKE statement. If in-line invocation is used, the result becomes in the sending item of the statement containing the method invocation.

11.4.3.9 Special Registers

In a method definition, you cannot reference the PROGRAM-STATUS (RETURN-CODE) special register.

Even if you set PROGRAM-STATUS to a specific value and execute a program end statement, no value will be stored in PROGRAM-STATUS for a calling program, when:

- A program is called from within a method definition,
- A program is called by the CALL statement having the RETURNING phrase, or
- The PROCEDURE DIVISION header of a program contains the RETURNING phrase.

11.4.3.10 Interfaces and Conformance

An object has an interface that contains method-names and parameter specifications supported by the object. A class defines interfaces for a factory object and an object.

When an object implements all the methods specified by the interface, the object can be used at any place where objects of that interface can be used. In this case, the object conforms to the interface. Refer to the topic "Conformance" of the "The Procedure Division" section later in this chapter for further details.

11.4.3.11 Polymorphism

Polymorphism lets one statement perform different things, based on context. In COBOL, a data item can contain objects of different classes. This capability dynamically binds method invocation on that data item to one of many methods. A method may be bound before execution. In general, however, a method is not bound until runtime.

Data items can be declared so that it could contain a given class or its subclass also.

11.4.3.12 Class Inheritance

A class can inherit from one or more classes. In this case, the inheriting class will have all the methods defined for the inherited classes. These include the methods inherited by the inherited class.

When a class inherits two or more classes, and when a class is going to inherit the methods carrying the same name from two or more classes, the classes to be inherited are inspected according to the order specified by the INHERITS clause, and the method that is found first is inherited.

The inheriting class inherits all the defined data from an inherited class, which includes data inherited by the inherited class. The inherited data is a part of each object of the inheriting class. But, they can only be referenced from the method defined by the class that has defined the data. The method-name defined in the inherited definition becomes a user-defined word for the inheriting definition, as if you declared the method-name in the inheriting definition in the relatively same position as declared in the inherited definition. The other user-defined words in the inherited definition are not inherited. These user-defined words can be used in the inheriting class as if they were not defined in the inherited class.

In addition, the interface of the inheriting class conforms to the interface of the inherited class. The interface of the inherited class will have all the method prototype definitions for the inherited class. These include the definitions inherited by the inherited class.

11.4.4 Life Cycle for Objects

An object is created as a result of the CREATE method invoked by a factory object. The CREATE method is invoked by the NEW method of the FJBASE class.

An object is deleted when:

1. The object finds that it cannot exist in a run unit any longer. But this is not true when:
    a. The object is referenced in the DATA DIVISION of a program in a run unit that is not canceled
    b. The object is referenced in the DATA DIVISION of a running method, or
    c. The object is referenced by another object or factory object.
2. The run unit is complete.

11.4.5 Life Cycle for Factory Objects

A factory object is created before it is referenced initially in a run unit, and deleted after the last reference by the run unit.

11.5 Identification Division and End Marker

This section describes the IDENTIFICATION DIVISION and end markers used in the object-oriented programming function.

11.5.1 Organization of the IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION identifies a program, class, factory object, object, and method. The format for the IDENTIFICATION DIVISION is shown below.

Format

```
[ IDENTIFICATION DIVISION. ]
{ PROGRAM-ID paragraph  }
{ CLASS-ID paragraph    }
{ FACTORY paragraph     }
{ OBJECT paragraph      }
{ METHOD-ID paragraph   }
[ AUTHOR paragraph      ]
```

[INSTALLATION paragraph]

[DATE-WRITTEN paragraph]

[DATE-COMPILED paragraph]

[SECURITY paragraph]

11.5.1.1 CLASS-ID Paragraph

The CLASS-ID paragraph indicates that the class definition begins with this IDENTIFICATION DIVISION, specifies the name to identify a class, and assigns a class attribute to a class.

Format

CLASS-ID. class-name-1 [AS literal-1]

[INHERITS {class-name-2} ...] .

Syntax Rules

1. Literal-1 must not be a figurative constant, but rather a nonnumeric literal or national language nonnumeric literal.
2. Class-name-2 must be the name of the class specified in the REPOSITORY paragraph of this source element.
3. Class-name-2 should not be the same as the name of a class declared in this class definition.
4. Class-name-2 should not be a name of special class.
5. Class-name-2 should not inherit directly or indirectly from class-name-1.

General Rules

1. Class-name-1 names the class declared in this class definition.

   When literal-1 is already specified, it becomes the external name of the class, and class-name-1 becomes the internal name. When literal-1 is not specified, class-name-1 becomes both the internal and external name.
2. The INHERITS clause specifies the name of the class inherited by class-name-1 according to the general rule described in the topic "Inheritance" above.
3. If the same name is inherited directly or indirectly two or more times, only one copy of the data of the class will be added to class-name-1.

11.5.1.2 FACTORY Paragraph

The FACTORY paragraph indicates that a factory definition begins with this IDENTIFICATION DIVISION.

Format

FACTORY.

11.5.1.3 OBJECT Paragraph

The OBJECT paragraph indicates that an object definition begins with this IDENTIFICATION DIVISION.

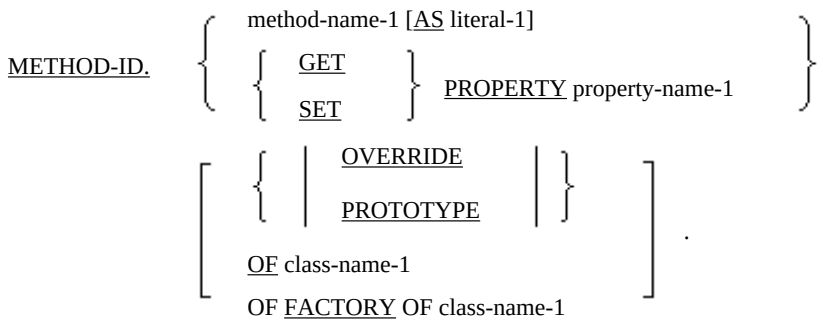
Format

OBJECT.

11.5.1.4 METHOD-ID Paragraph

The METHOD-ID paragraph indicates that a method definition begins with this IDENTIFICATION DIVISION. This paragraph also specifies a name to identify a method and assigns a method attribute to the method.

Format



Syntax Rules

1. Literal-1 must not be a figurative constant, but rather a nonnumeric literal or national language nonnumeric literal.
2. Class-name-1 must be the name of the class specified in the REPOSITORY paragraph of this source element. The class identified by class-name-1 must include a method prototype that has the same external name and interface as the method declared in this method definition.
3. If the PROTOTYPE phrase is specified as the METHOD-ID paragraph included in a factory definition, a separate method definition with the same name as the method declared in this method definition must exist, which specifies the FACTORY phrase of a class including this method definition.
4. If the PROTOTYPE phrase is specified as the METHOD-ID paragraph included in an object definition, a separated method definition with the same name as the method declared in this method must exist, which specifies the class including this method definition.
5. If the PROTOTYPE clause is specified, the DATA DIVISION of this method definition must contain only a LINKAGE SECTION, and the PROCEDURE DIVISION must contain only the PROCEDURE DIVISION header.
6. If class-name-1 is specified, this method definition should not be included in a class definition. class-name-1 must be the name of the class containing a method prototype with the same external name as the method declared in this method definition. If the FACTORY phrase is specified, the corresponding method prototype must be declared in the factory definition. If the FACTORY clause is not specified, the corresponding method prototype must be declared in the object definition.
7. If class-name-1 is specified, this method behaves in the same manner as it does when an associated method prototype is replaced by this method definition.
8. If class-name-1 is not specified, this method definition must be included in a class definition.
9. If the OVERRIDE clause is specified, the inherited method with the same external name as the method declared in this method definition must be defined in the inherited class.
10. If the OVERRIDE clause is not specified, no inherited method with the same external name as the method declared in this method definition must be defined in any inherited class.
11. If property-name-1 is specified as a data-name in the WORKING-STORAGE SECTION of the factory definition or object definition including this, the data description entry with that data-name should not have the PROPERTY clause.
12. If the GET phrase is specified, the PROCEDURE DIVISION header should not contain the USING phrase. It should have the RETURN clause.
13. If the SET phrase is specified, one USING parameter should be specified in the PROCEDURE DIVISION header. It should not have the RETURNING phrase.

General Rules

1. method-name-1 names a method declared in this method definition.

   When literal-1 is already specified, it becomes the external name of the method, and method-name-1 becomes the internal name. When literal-1 is not specified, method-name-1 becomes both the internal and external name.

2. The PROTOTYPE clause indicates that this is a method prototype.
3. The OVERRIDE clause indicates that this method will replace the inherited method.
4. The external name of the method can be used in method invocation using an object identifier that references the object containing this method.
5. If the external name of the method is the same as the method-name inherited by the definition including it, the parameter definition in the PROCEDURE DIVISION header must comply with the conformance rules to assure that the definition including this method definition conform to each inherited definition, based on topic "Conformance Between Interfaces" of the "Procedure Division" section of this chapter.
6. If a user-defined word (for example, a data name) with the same name is defined in the method definition and in the factory definition or object definition containing the method definition, the word used in this method will reference the declarative within this method in this method definition. Declaratives in the factory that include the method definition or in the object definition cannot be referenced from this method.
7. If the GET phrase is specified, this method will be a get property method with property-name-1.
8. If the SET phrase is specified, this method will be a set property method with property-name-1.

11.5.1.5 AUTHOR Paragraph

Refer to the section "Composition of the Identification Division" of Chapter 3, "Identification Division and End Program Header" for details of the rules not native to the object-oriented programming function.

Syntax Rules

The AUTHOR paragraph can only be specified in the IDENTIFICATION DIVISION of a program definition.

11.5.1.6 INSTALLATION Paragraph

Refer to the section "Composition of the Identification Division" of Chapter 3, "Identification Division and End Program Header" for details of the rules not native to the object-oriented programming function.

Syntax Rules

The INSTALLATION paragraph can only be specified in the IDENTIFICATION DIVISION of a program definition.

11.5.1.7 DATE-WRITTEN Paragraph

Refer to the section "Composition of the Identification Division" of Chapter 3, "Identification Division and End Program Header" for details of the rules not native to the object-oriented programming function.

Syntax Rules

The DATE-WRITTEN paragraph can only be specified in the IDENTIFICATION DIVISION of a program definition.

11.5.1.8 DATE-COMPILED Paragraph

Refer to topic "DATE-COMPILED Paragraph" of the "Composition of the Identification Division" section of Chapter 3, "Identification Division and End Program Header" for details of the rules not native to the object-oriented programming function.

Syntax Rules

The DATE-COMPILED paragraph can only be specified in the IDENTIFICATION DIVISION of a program definition.

11.5.1.9 SECURITY Paragraph

Refer to the section "Composition of the Identification Division" of Chapter 3, "Identification Division and End Program Header" for details of the rules not native to the object-oriented programming function.

Syntax Rules

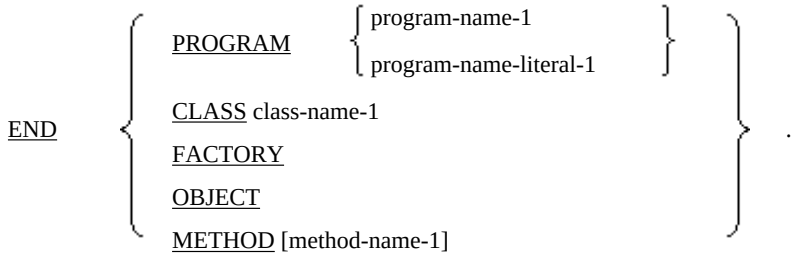
The SECURITY paragraph can be specified only in the IDENTIFICATION DIVISION of a program definition.

11.5.2 END Marker

This section describes the END marker.

Refer to topic "The End Program Header" of the "Composition of the Identification Division" section of Chapter 3, "Identification Division and End Program Header" for details of the format of the END PROGRAM clause.

Format



Syntax Rules

The END marker should be specified in a source unit that includes another source unit, one included in another, or one followed by another.

General Rules

1. The END marker indicates the end of the specified source unit.
2. If the source unit terminated by the END marker is included in another source unit, the next statement must be the first statement of the source unit, or the END marker that terminates the including source unit.
3. If the source unit terminated by the END header is not included in another source unit, the next statement must be the first statement of a source unit to be compiled separately.

11.5.2.1 The END CLASS Header

This section describes the END CLASS header.

Format

END CLASS class-name-1.

Syntax Rules

Class-name-1 must be the same as the class-name specified in the corresponding CLASS-ID paragraph. Refer to the topic "The END Marker" above for other applicable syntax rules.

General Rules

Refer to the topic "The END Marker" above for general rules.

11.5.2.2 The END FACTORY Header

This section describes the END FACTORY header.

Format

END FACTORY.

Refer to the topic "The END Marker" above for coding and general rules.

11.5.2.3 The END OBJECT Header

This section describes the END OBJECT header.

Format

END OBJECT.

Refer to the topic "The END Marker" above for coding and general rules.

11.5.2.4 The END METHOD Header

This section describes the END METHOD header.

Format

END METHOD [method-name-1].

Syntax Rules

Method-name-1 must be the same as the method-name specified in the corresponding METHOD-ID paragraph. If the PROPERTY clause is specified in the METHOD-ID paragraph, method-name-1 should be omitted.

Refer to the topic "The END Marker" above for other syntax rules.

General Rules

Refer to the topic "The END Marker" above for general rules.

11.6 The ENVIRONMENT DIVISION

This section describes the CONFIGURATION SECTION and INPUT-OUTPUT SECTION native to the object-oriented programming function.

11.6.1 The CONFIGURATION SECTION

This section describes the format for the CONFIGURATION SECTION using the object-oriented programming function. Refer to the topic "Configuration Section" of the "Composition of the Environment Division" section of Chapter 4, "Environment Division" for other rules applicable to the CONFIGURATION SECTION.

Format

CONFIGURATION SECTION.

[SOURCE-COMPUTER paragraph]

[OBJECT-COMPUTER paragraph]

[SPECIAL-NAMES paragraph]

[REPOSITORY paragraph]

Syntax Rules

1. The CONFIGURATION SECTION can be specified in a class definition, program definition, or method definition. However, it cannot be specified in a factory definition and object definition. In addition, the CONFIGURATION SECTION cannot be specified in a method definition contained in a class definition.
2. The SOURCE-COMPUTER and OBJECT-COMPUTER paragraphs cannot be coded in the CONFIGURATION SECTION of a class or method definition.

3. The SPECIAL-NAMES paragraph cannot be coded in the CONFIGURATION SECTION of a method definition.

11.6.1.1 The REPOSITORY Paragraph

The REPOSITORY paragraph specifies a class-name to be explicitly or implicitly used within the scope of table of this ENVIRONMENT DIVISION (See the Note below). The REPOSITORY paragraph can be coded in a class, method, and program definitions.

Note

That is, within a source unit that contains this ENVIRONMENT DIVISION or a source unit contained in it.

Format

REPOSITORY.

[{ class-specifier / property-specifier } ... .]

Class-Specifier

Format

CLASS class-name-1
[AS literal-1]

Property-Specifier

Format

PROPERTY property-name-1
[AS literal-2]

Syntax Rules

1. Class-name-1 can be specified in the REPOSITORY paragraph only once.
2. Literal-1 must not be a figurative constant, but rather a nonnumeric literal or national language nonnumeric literal.
3. When you need to reference the data item declared in a class definition from a method definition not included in the class definition, you must declare the class-name in the REPOSITORY paragraph of the method definition of the class name, if the data item has an entry with the class-name. In addition, the declarative must be exactly the same as that in the REPOSITORY paragraph of the related class definition except for the internal name.
4. If a REPOSITORY paragraph is specified in a class definition, class-name-1 must not be the class-name given to that class.
5. The property-specifier declares a name that can be specified by an object property within the scope of the ENVIRONMENT DIVISION. However, it can be omitted with this implementation of COBOL.
    a. Specifying literal-2 means that an alias is assigned. Information on the property associated with the data in literal-2 must be included in the class declared in the REPOSITORY paragraph.
    b. When literal-2 is not specified, information on the property associated with property-name-1 must be included in the class declared in the REPOSITORY paragraph.

General Rules

1. class-name-1 is a name of the class to be used within the scope of the ENVIRONMENT DIVISION including it.

2. If literal-1 is specified, class-name-1 indicates the internal name, and literal-1 indicates the external name. However, when the value of literal-1 is any of the following, the class that is identified by class-name-1 is a special class.

   - "*COM" (or "*OLE")
   - "*COM-ARRAY" (or "*OLE-ARRAY")
   - "*COM-EXCEPTION" (or "*OLE-EXCEPTION")
   - "*COB-BINDTABLE"
   - "*COM: COM server name: COM class name"
     - COM server name:

       Arbitrary name used for association with a type library
     - COM class name:

       dispinterface name or coclass name

3. Special classes are divided into the following two types according to the mode of binding an invoked method and the function available for this operation:

   a. Early-bind special class
      Binding and conformance checking is implemented during compilation. Therefore, the execution-time performance is better than late-bind classes. Method in-line invocation, object properties, and parameters with the BY CONTENT phrase can be used.
      An early-bind special class is indicated when the value of literal-1 is the following:

      - "*COM: COM server name: COM class name"

   b. Late-bind special class
      Binding and conformance checking is implemented during execution. Method in-line invocation, object properties, and parameters with the BY CONTENT phrase cannot be used.
      A late-bind special class is indicated when the value of literal-1 is one of the following:

      - "*COM" (or "*OLE")
      - "COM-ARRAY" (or "*OLE-ARRAY")
      - "*COM-EXCEPTION" (or "*OLE-EXCEPTION")
      - "*COB-BINDTABLE"

Note

These functions - except "*COB-BINDTABLE" - are specific to [Win32]. "*COB-BINDTABLE" class cannot be used for [Winx64], [Linux], [LinuxIPF] and [Linux64].

11.6.2 The INPUT-OUTPUT SECTION

This section describes the form for the INPUT-OUTPUT SECTION using the object-oriented programming function. Refer to the topic "The Input-Output SECTION" of the "Composition of the Environment Division" section of Chapter 4, "Environment Division" for other rules.

Syntax Rules

The INPUT-OUTPUT SECTION can be coded in a program, factory, object, and method definitions, but not in a class definition.

11.6.2.1 The I-O-CONTROL Paragraph

This section describes the rules for the I-O-CONTROL paragraph applicable to the object-oriented programming function. Refer to the topic "I-O-Control Paragraph (I-O-CONTROL)" of the section "Composition of the Environment Division" of Chapter 4, "Environment Division" for details of the other rules.

Syntax Rules

1. In the I-O-CONTROL paragraph of a factory, object, or method definitions, you cannot use the APPLY, MULTICONVERSATION-MODE, APPLY SAVED-AREA, or MULTIPLE FILE TAPE clauses.
2. The USAGE OBJECT REFERENCE clause cannot be used for data-name-1 in the APPLY SAVED-AREA clause or a data item subordinate to data-name-1.

11.7 The DATA DIVISION

This section describes the rules for the DATA DIVISION native to the object-oriented programming function.

11.7.1 The FILE SECTION

This section describes the form for the FILE SECTION using the object-oriented programming function.

Refer to section "Composition of the Data Division" of Chapter 5, "Data Division" for other rules for the FILE SECTION.

Syntax Rules

1. The FILE SECTION can be specified in a program definition. In a file definition, the FILE SECTION can be coded only in a factory, object, or method definition.
2. In the FILE SECTION of a factory, object, or method definition, the LABEL RECORD, VALUE OF, and DATA RECORD clauses cannot be used.

11.7.2 The LINKAGE SECTION

This section describes the form for the LINKAGE SECTION using the object-oriented programming function. Refer to the section "Composition of the Data Division" of Chapter 5, "Data Division" for other rules for the LINKAGE SECTION.

Syntax Rules

1. A data item defined in the LINKAGE SECTION of a source element can be referenced in the PROCEDURE DIVISION of that source element, when:
    a. The data item is the operand of the USING or RETURNING phrase of the PROCEDURE DIVISION header,
    b. The data item is subordinate to the operand of the USING or RETURNING phrase of the PROCEDURE DIVISION header,
    c. The REDEFINES or RENAMES phrase has been applied to the data item which meets the two conditions described in (a) and (b),
    d. The data item is subordinate to an item which meets the condition described in (c), or
    e. The data item is a condition-name or index-name associated with a data item which meets one of the above conditions.

11.7.3 File Description Entries

This section describes file description entries using the object-oriented programming function. Refer to "File Description Entries" for details.

11.7.3.1 The LINAGE Clause

Refer to the topic "LINAGE Clause (Sequential File)" of the "File Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Syntax Rules

1. In the DATA DIVISION of a factory or object division, the LINAGE clause cannot be used.
2. In a method description, the LINAGE clause cannot be used to describe a file for which the EXTERNAL clause has been specified.

11.7.4 Data Description Entries

This section describes the form for data description entries using the object-oriented programming function.

Refer to the section "Data Description Entry" of Chapter 5, "Data Division" for other rules of clauses not shown in Format 1.

Format 1

Defining data-names

Level-number [data-name-1 | FILLER]

[REDEFINES clause]

[TYPEDEF clause]

[BASED ON clause]

[BLANK WHEN ZERO clause]

[CHARACTER TYPE clause]

[EXTERNAL clause]

[GLOBAL clause]

[JUSTIFIED clause]

[OCCURS clause]

[PICTURE clause]

[PRINTING POSITION clause]

[PROPERTY clause]

[SIGN clause]

[SYNCHRONIZED clause]

[USAGE clause]

[TYPE clause]

[VALUE clause]

[ANY LENGTH clause].

11.7.4.1 The ANY LENGTH clause

The ANY LENGTH clause specifies that the length of the data item of the LINKAGE SECTION is determined during at execution time.

Format

ANY LENGTH

Syntax Rules

1. The ANY LENGTH clause can be specified in a 01 item that is specified in the LINKAGE SECTION of the method definition, except for the property method.
2. The ANY LENGTH clause can be specified in the item in which only the PICTURE clause is specified.
   The character string of the PICTURE clause must be one character, either "X" or "N".
3. The data description entry of level number 66 and the condition-name description entry must not be defined immediately after the data description entry in which the ANY LENGTH clause is specified.

General Rules

1. The length of the specified data item of the ANY LENGTH clause becomes the same length as the corresponding argument item.
2. Data-name-1 can be specified in the PROCEDURE DIVISION except at the following locations.
   - The USING or RETURNING phrases of a CALL statement
   - The RETURNING phrases of an INVOKE statement
   - The INITIALIZE statement
   - As the argument of an in-line invocation
   - As the argument of the following intrinsic functions
     - FUNCTION UPPER-CASE
     - FUNCTION LOWER-CASE
     - FUNCTION REVERSE
     - FUNCTION MAX
     - FUNCTION MIN
     - FUNCTION CAST-ALPHANUMERIC
     - FUNCTION NATIONAL
     - FUNCTION UCS2-OF
     - FUNCTION UTF8-OF

11.7.4.2 The CHARACTER TYPE clause

Refer to topic "CHARACTER TYPE Clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Syntax Rules

1. In the DATA DIVISION of a factory or object definition, the CHARACTER TYPE clause cannot specify the DEPENDING ON phrase.
2. In the LINKAGE SECTION of a method prototype definition, the CHARACTER TYPE clause cannot be used.

11.7.4.3 The EXTERNAL Clause

Refer to the topic "EXTERNAL Clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Syntax Rules

1. The EXTERNAL clause must not be specified in the DATA DIVISION of a factory or object definition.

2. The EXTERNAL clause should not be used for a data item to be used for the object reference or for the group item that includes the object reference data item.

11.7.4.4 The GLOBAL Clause

Refer to the topic "GLOBAL Clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Syntax Rules

The GLOBAL clause must not be specified in the DATA DIVISION of a factory or object definition.

11.7.4.5 The OCCURS Clause

Refer to the topic "OCCURS Clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Syntax Rules

A data item of which class is an object cannot specify the KEY IS phrase.

11.7.4.6 The PRINTING POSITION Clause

Refer to the topic "PRINTING POSITION Clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Syntax Rules

The PRINTING POSITION clause cannot be specified in the LINKAGE SECTION of a method prototype definition.

11.7.4.7 The PROPERTY Clause

The PROPERTY clause indicates that this data item is assumed to be this object's property and generates the GET or SET method.

Format

Syntax Rules

1. The PROPERTY clause can only be specified in the WORKING-STORAGE SECTION of a factory or object definition.
2. The PROPERTY clause cannot be specified for a data item using the OCCURS clause or a data item subordinate to that data item.
3. The PROPERTY clause cannot be specified for a data item in which the TYPEDEF clause is specified, or for the data item that depends on it.
4. The PROPERTY clause can only be used for elementary items that do not need to be qualified to uniquely reference a name.
5. The PROPERTY clause cannot be used for a data item for which the USAGE OBJECT REFERENCE including the SELF phrase has been specified.
6. The PROPERTY clause cannot be used for a data description entry to be used as a pointer.
7. When you want to reference the PROPERTY clause and the TYPE clause at the same time, the type to which the TYPE clause refers must be defined as the basic items that do not infringe on syntax rules (4) or (5).

General Rules

1. If the GET phrase is not specified, the PROPERTY clause will generate a method defined for a factory or object including it.

   If a data item is used as an object reference or index, this method's implicit definition will be as follows:

   The "data-name" will be replaced by a property name.

```
METHOD-ID. GET PROPERTY data-name.
DATA DIVISION.
  LINKAGE SECTION.
  01 LS-data-name  data description.
 PROCEDURE DIVISION RETURNING LS-data-name.
 paragraph-name.
     SET LS-data-name TO data-name
     EXIT METHOD.
 END METHOD.
```

   If the class of a data-item is alphanumeric or national, this method's implicit definition will be as follows:

```
METHOD-ID. GET PROPERTY data-name.
DATA DIVISION.
  LINKAGE SECTION.
  01 LS-data-name  data description.
 PROCEDURE DIVISION RETURNING LS-data-name.
 paragraph-name.
     MOVE data-name TO LS-data-name(1:)
     EXIT METHOD.
 END METHOD.
```

   In cases other than those above, the implicit definition of the method will be as follows:

```
METHOD-ID. GET PROPERTY data-name.
DATA DIVISION.
  LINKAGE SECTION.
  01 LS-data-name  data description.
 PROCEDURE DIVISION RETURNING LS-data-name.
 paragraph-name.
     MOVE data-name TO LS-data-name
     EXIT METHOD.
 END METHOD.
```

   Here, LS-data-name has the same data description entries as this data item, including subordinate clauses, except for the following clauses:

   - INDEXED BY clause
   - PROPERTY clause
   - VALUE clause
   - REDEFINES clause
   - CHARACTER TYPE clause
   - PRINTING POSITION clause

2. If the SET phrase is not specified, the PROPERTY clause generates a method defined for a factory or object including it.

   If a data item is used as an object reference, the implicit definition of the method will be as follows: "data-name" will be replaced by a property name.

```
METHOD-ID. SET PROPERTY data-name.
DATA DIVISION.
  LINKAGE SECTION.
```

```
 01 LS-data-name  data description.
PROCEDURE DIVISION USING LS-data-name.
paragraph-name.
    SET data-name TO LS-data-name
    EXIT METHOD.
END METHOD.
```

If the class of a data-item is alphanumeric or national, the implicit definition of the method will be as follows:

```
METHOD-ID. SET PROPERTY data-name.
DATA DIVISION.
  LINKAGE SECTION.
  01 LS-data-name  data description.
PROCEDURE DIVISION USING LS-data-name.
paragraph-name.
    MOVE LS-data-name TO data-name(1:)
    EXIT METHOD.
END METHOD.
```

In cases other than the above, the implicit definition of the method will be as follows:

```
METHOD-ID. SET PROPERTY data-name.
DATA DIVISION.
  LINKAGE SECTION.
  01 LS-data-name  data description.
PROCEDURE DIVISION USING LS-data-name.
paragraph-name.
    MOVE LS-data-name TO data-name
    EXIT METHOD.
END METHOD.
```

Here, LS-data-name has the same data description entries as this data item, including subordinate clauses, except for the following clauses:

- INDEXED BY clause
- PROPERTY clause
- VALUE clause
- REDEFINES clause
- CHARACTER TYPE clause
- PRINTING POSITION clause

11.7.4.8 The REDEFINES Clause

Refer to the topic "REDEFINES Clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Format

Level-Number [data-name-1 / FILLER] REDEFINES data-name-2

Syntax Rules

1. The REDEFINES clause cannot be used for a data item declared as an object reference.
2. The data description entry for data-name-2 should not include a data item to be used as an object reference.
3. Data-name-2 must not have the ANY LENGTH clause specified.

11.7.4.9 The RENAMES clause

Refer to the "RENAMES Clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Syntax Rules

1. Data-name-2 and data-name-3 must not be data items which specify the ANY LENGTH clause. Also, the items in the scope of data-name-2 to data-name-3 must not contain data items which specify the ANY LENGTH clause.
2. Data-name-2 and data-name-3 must not be data items that have declared that their application is an object reference. These data items must not be contained in the scope of data-name-2 to data-name-3.

11.7.4.10 The TYPE clause

Refer to the topic "TYPE Clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Syntax Rules

The TYPE clause cannot be specified in the LINKAGE section for property method definition.

11.7.4.11 The TYPEDEF clause

Refer to the topic "TYPEDEF Clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Syntax Rules

When STRONG is specified, a subordinate item must not contain an object reference data item with the SELF phrase.

11.7.4.12 The USAGE clause

Refer to the topic "USAGE clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

11.7.4.12.1 The USAGE IS INDEX clause

Format

[USAGE IS] INDEX

Syntax Rules

Basic items that have specified the USAGE IS INDEX clause, and basic items that depend on a group item specifying the USAGE IS INDEX clause can only be used in the following positions.

- SEARCH statement
- SET statement
- DISPLAY statement
- Relation condition
- PROCEDURE DIVISION header or USING phrase of the ENTRY statement
- USING phrase of the CALL or INVOKE statement

11.7.4.12.2 The USAGE IS OBJECT REFERENCE clause

Refer to the topic "USAGE Clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Format

```
[USAGE IS]
  OBJECT REFERENCE
[ [ [ { FACTORY } OF ] SELF ]
      { CLASS   }
  [FACTORY OF] class-name-1 [ONLY] ]
```

Syntax Rules

1. The BLANK WHEN ZERO, JUSTIFIED, PICTURE, or SYNCHRONIZED clauses cannot be used for a data item to be used as an object reference.
2. An elementary item whose class is object must not be a conditional variable.
3. The USAGE OBJECT REFERENCE clause must not be specified for data description entries of a group item.
4. The USAGE OBJECT REFERENCE clause can only be specified in the WORKING-STORAGE SECTION and LINKAGE SECTION.
5. The USAGE OBJECT REFERENCE clause cannot be specified for a data item that also includes the OCCURS DEPENDING ON clause or a data item subordinate to that data item.
6. The USAGE OBJECT REFERENCE clause cannot be used for a data item located in a variable position within a record.
7. Class-name-1 must be either a class name that has declared the class-specifier in the source unit including the data description entry or a class-name of class definition including the data description entry.
8. The USAGE OBJECT REFERENCE SELF clause can only be used for a factory or factory method description.
9. The USAGE OBJECT REFERENCE FACTORY OF SELF clause can only be used for an object or object method description.
10. The USAGE OBJECT REFERENCE CLASS OF SELF clause can only be used for an object or object method description.
11. If class-name-1 is a name of special class, no other optional phrase can be specified.

General Rules

1. A data description including the USAGE OBJECT REFERENCE clause assigns enough area to hold object references.
2. A data description entry specifying the USAGE OBJECT REFERENCE clause describes an object reference identifier to be used to reference an object.
3. The content of a data description entry specifying the USAGE OBJECT REFERENCE clause contains a reference to an object. Refer to Table 11.2 "Options for USAGE OBJECT REFERENCE" for the additional rules for the options you can choose.

Table 11.2 "Options for USAGE OBJECT REFERENCE

Selected Data Description Entry	Objects to be referenced
SELF	An object created by a factory object identified by SELF, or NULL object
FACTORY OF SELF	A factory object identified by SELF, or NULL object
CLASS OF SELF	An object created by a factory object which created the object identified by SELF, or NULL object
class-name-1	An object of class-name-1 or a class inheriting from class-name-1, or NULL object
class-name-1 ONLY	An object of class-name-1 or NULL object

Selected Data Description Entry	Objects to be referenced
FACTORY OF class-name-1	A factory object of class-name-1 or a class inheriting from class-name-1, or NULL object
FACTORY OF class-name-1 ONLY	A factory object of class-name-1 or NULL object

Note

When you have specified the USAGE OBJECT REFERENCE clause to an item that depends on a weakly-typed group item, referencing the group item is limited in order to guarantee the validity of the object reference data item.

A group item can be referenced in the following cases only.

- LENGTH function
- LENG function

11.7.4.13 The VALUE Clause

Refer to the topic "VALUE Clause" of the "Data Description Entry" section of Chapter 5, "Data Division" for rules not native to the object-oriented programming function.

Syntax Rules

The VALUE clause cannot be used for a data item to be used as an object reference for the group item including it.

General Rules

A data item of whose category is object reference is initialized to reference a NULL object. This initial value is set at the time the VALUE clause takes effect.

11.8 The PROCEDURE DIVISION

This section describes PROCEDURE DIVISION rules that are native to the object-oriented programming function.

11.8.1 Composition of the PROCEDURE DIVISION

The PROCEDURE DIVISION in a method contains procedures to be executed.

The PROCEDURE DIVISION of an object or factory definition contains the method to be invoked on the object or factory object.

Format 1

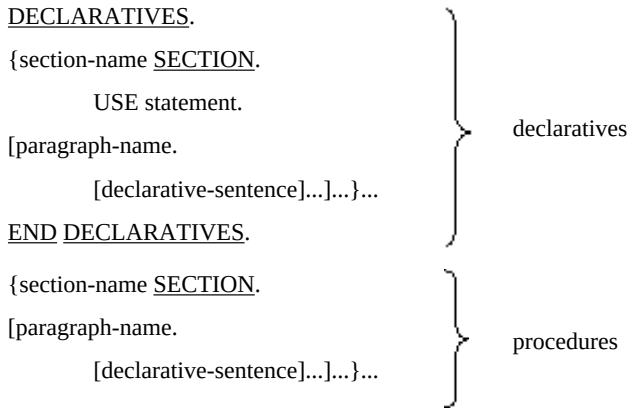
No declaratives are included.

```
PROCEDURE DIVISION.
      {paragraph-name.                  }
      [declarative-sentence]...}...     } Procedures
```

Format 2

Declaratives are included.

```
PROCEDURE DIVISION.
```



Format 3

The PROCEDURE DIVISION of an object or factory definition

```
PROCEDURE DIVISION.
[{method definition}...]
```

In this implementation of COBOL, both section-names for procedures and paragraph-names can be omitted.

Imperative Statements

Imperative Statements are:

- INVOKE
- RAISE
- EXIT METHOD
- EXIT PROGRAM
- USE

11.8.2 PROCEDURE DIVISION Header

This section describes the rules for the PROCEDURE DIVISION header using the object-oriented programming function.

Refer to the section "Procedure Division Header" of Chapter 6, "Procedure Division" for other rules applicable to the PROCEDURE DIVISION header.

Format 1

The PROCEDURE DIVISION of a method or method prototype definition

```
PROCEDURE DIVISION
        [USING {data-name-1}...]
        [RETURNING data-name-2]
        [RAISING {class-name-1} ...].
```

Format 2

The PROCEDURE DIVISION of an object or factory definition

```
PROCEDURE DIVISION.
[{method definition}...]
```

Syntax Rules

- Rules for Format 1

  1. If data-name-1 is defined with the USAGE OBJECT REFERENCE clause, the SELF, FACTORY OF SELF, or CLASS OF SELF phrases should not be used.
  2. In a property method with the SET phrase, data-name-1 must not be a strongly typed group item.
  3. Data-name-2 should not be the same as data-name-1.
  4. Data-name-2 should not be a pointer data item.
  5. In a property method with the GET phrase, data-name-2 must not be a strongly typed group item.
  6. Class-name-1 must be the name of the class that is specified for the REPOSITORY paragraph.
  7. Class-name-1 must not be a special class name except for *COM-EXCEPTION (or *OLE-EXCEPTION).

- Rules for Format 2

  The format 2 can be used only for factory and object definitions.

General Rules

1. During execution, the first statement in the PROCEDURE DIVISION, except for declaratives, is executed first. Statements are executed in the order in which they appeared at compile time.
2. The USING phrase identifies the name of a formal parameter to be used by a method or program. Arguments to be passed to it are identified by:

   - The USING phrase of the INVOKE statement of the invoking source element, or
   - The operand specified at in-line invocation of a method.

   Formal parameters and arguments are associated with each other based on their positions.
3. Data-name-1 is a formal parameter for a method or a program.
4. Data-name-2 is a name to be used within a method or a program to return a result to the invoking element in compliance with topics "Method Results" of the "Compilation Group Structure" section earlier in this chapter and "PROGRAM-STATUS and RETURN-CODE Special Registers" of the "Inter-program Communication Module" section of Chapter 2, "COBOL Modules".
5. The initial values for the returning item, data-name-2, is undefined.
6. If arguments are passed, the invoked program will handle them as if the data items in the LINKAGE SECTION occupied the same area as the arguments of the invoking program.
7. If arguments are passed by the BY CONTENT phrase, the invoked program will handle them as if the area of the LINKAGE SECTION was allocated by the invoking program upon invocation. This record does not occupy the same area as the arguments of the invoking program.

   For its length, the assigned record has exactly the same alphanumeric character positions as the record in the LINKAGE SECTION.

   The following statements are executed for the argument as a sending item, and for the allocated record having a description coded in the LINKAGE SECTION as a receiving item.

   - The COMPUTE statement if the formal parameter is numeric
   - The SET statement if the formal parameter is an index data item, or if it is to be used as an object reference.
   - MOVE statement cases other than the above

   The allocated record is handled as if it was an argument and had been passed by the BY REFERENCE clause.
8. In the invoked program, the references by data-name-1 and data-name-2 are always resolved by the description in the LINKAGE SECTION.
9. The class-name of an exception class that can occur in the EXIT statement must be specified as class-name-1.

11.8.3 Common Rules Applicable to Statements

This section describes the common rules applicable to statements for the object-oriented programming function, as well as the phrases which can be used in the statements.

11.8.3.1 Invocation Operator

The invocation operator is two consecutive COBOL characters "::" following a separator, a blank. These two colons should be enclosed by blanks. Refer to the topic "In-Line Method Invocation" of the "General Rules" section earlier in this chapter for the use of invocation operators.

11.8.3.2 Relation Condition between Object Identifiers

An object reference identifier can be compared with another object reference identifier or a predefined object identifier described in the topic "Predefined Object Identifier" of the "General Rules" section earlier in this chapter.

- Use either the relational operator [NOT] EQUAL or [NOT] = to compare object reference identifiers.
- For object reference identifiers, the relation "object-identifier-1 = object-identifier-2" holds the value of true only when the object identified by the object-identifier-1 is the same as the object identified by the object-identifier-2.
- The predefined object identifier NULL cannot be compared with itself.
- A class-name of special class cannot be compared.

11.8.4 Execution

This section describes execution.

11.8.4.1 Status of Methods and Objects

This section describes the status of methods and objects.

11.8.4.1.1 Status of Methods

The status of a method is either active or inactive at any time during execution. A method returns to the initial state every time it is activated.

Active State

The entity of a method becomes active if method invocation is successful. It remains active until the STOP statement or the explicit or implicit EXIT METHOD statement in the entity of the method is executed.

A method becomes active recursively. However, each entity of the method gets activated only once.

The control mechanism of a PERFORM statement contained in a method returns to the initial state every time the method is activated.

Initial State of Data

Data within a method returns to the initial state every time the method in which the data was defined is activated.

When data within a method are in the initial state,

1. Internal data described in the WORKING-STORAGE SECTION will be initialized in the following manner:

   If the VALUE clause is used to define data entries, they will be initialized with defined values. If not, their initial values will be undefined. Initial values for indices will also be undefined.

2. Internal file connectors in a method are initialized by setting the state of files to the closed state.

11.8.4.1.2 State of Data Associated Only with an Object

- The initial state of an object is the state of the object immediately after it is created.

- Internal data defined in the WORKING-STORAGE SECTION of an object is initialized in the same manner as internal data described in the WORKING-STORAGE SECTION of a method or program.
- Files that have internal file connectors associated with an object are in the closed state.

11.8.4.2 Explicit and Implicit Control Flow

When there are no more executable statements, and control does not move out of a source unit, the flow of control is undefined, unless the following conditions are met:

1. For a program, if statements other than declaratives are being executed and the program is under control of a CALL statement, then an implicit EXIT PROGRAM statement will be executed.
2. For a method, if statements other than declaratives are being executed, an implicit EXIT METHOD statement with no other optional phrases specified will be executed.

11.8.4.3 Manipulation of Conditions

This section describes exception conditions and exception objects.

11.8.4.3.1 Exception Conditions

Exception conditions are the conditions associated with exception objects.

11.8.4.3.2 Exception Objects

By using exception objects, you can use the USE statement to handle exceptions.

Two types of objects become exception objects. One is the object of a user-defined class, and one is the object of special class *COM-EXCEPTION (or *OLE-EXCEPTION).

The exception object of the user-defined class occurs by specifying the object to the RAISE or the EXIT statement. Any objects can be specified, except the object of a special class. Refer to the topics "EXIT statement (Object-Oriented Programming)" and "RAISE statement (Object-Oriented Programming)" later in this section for more details.

The exception object of the special class *OLE-EXCEPTION occurs automatically when an exception occurs in the OLE linkage. The exception object of the special class *OLE-EXCEPTION cannot be generated freely by users.

When an exception object occurs, one of the following will occur.

- The associated declaratives will be executed, if the class of the exception object or the class from which that class inherits is specified in the USE statement within the program. However, the declaratives will not be executed if the exception object is a factory object. If the declaratives are successfully executed (see the note below), execution continues in the same manner as a normal execution.

  That is, in the following cases:

  - There is no EXIT PROGRAM statement.
  - There is no EXIT METHOD statement.
  - There is no STOP RUN statement.
  - No program or method to terminate the run unit is invoked.
- If there is no appropriate USE statement in the source unit (see the note below), the source unit will terminate abnormally.

  For the RAISE statement, if there is no appropriate USE statement in the source unit, no exception object takes place. Refer to the topic "The RAISE Statement (Object-Oriented Programming) later in this section for further details.

11.8.5 Conformance

Conformance is a unilateral relationship from one interface to another or from an object to an interface.

The rules of conformance apply to data items declared as object references in the following cases:

- Assign into a data item

- Invocation of a method on a data item
- A data item is used as a parameter for a method
- A data item is used as a returning item from method invocation

In the following cases, the conformance rules also apply to data items other than object references:

- A data item is used as a parameter for a method
- A data item is used as a returning item from method invocation

11.8.5.1 Interfaces of a Class

Every class has two interfaces. One is an object's interface that is configured by the definitions of all the object methods including inherited ones. Another is a factory object's interface, which is configured by the definitions of all the factory methods including inherited ones.

11.8.5.2 Interfaces of an Object

Every object has its corresponding interface (object interface or factory object interface).

11.8.5.3 Conformance between Interfaces

Conformance is the relationship between two interfaces.

Interface-1 conforms to interface-2 if the condition described in either (1) or (2) below is met:

1. Interface-1 is identical to interface-2.
2. For each method of interface-2, interface-1 has a method with the same name and same number of parameters which satisfies the conditions described in (a) through (b) below:
    a. For each parameter with each method-name of interface-2, the declarative of the parameter of interface-1 conforms to that of interface-2, and vice versa.
    b. For each method-name of interface-2, one of the following conditions is satisfied:
        - If the USAGE OBJECT REFERENCE clause with the SELF phrase is not used for the declarative of a returning item from interface-1, the declarative of the returning item from interface-1 conforms to that from interface-2, and vice versa.
        - If the USAGE OBJECT REFERENCE clause with the SELF phrase is used for the declarative of a returning item from interface-1, the declarative of the USAGE clause for the returning item from interface-1 is exactly the same as that from interface-2.

11.8.5.4 Conformance for Assignment

Operations to assign data declared as object references must follow the conformance rules. The interface of a sending item must conform to the declarative for the receiving item in compliance with the rules described in the topics "Interfaces of an Object" and "Conformance Between Interfaces" above. These rules apply to assignment using the SET statement and BY CONTENT phrase to pass parameters.

Refer to Table 11.3 "Conformance for Assignment (except for Object Property, In-Line Invocation of a Method, and Special Class)," Table 11.4 "Conformance for Assignment (Object Property & In-Line Invocation of a Method)," and Table 11.5 "Conformance for Assignment (Special Class)" below for details.

Table 11.3 Conformance for Assignment (except for Object Property, In-Line Method Invocation, and Special Class)

Sending area	Receiving area							
	class-name		FACTORY class-name		UNIVERSAL	SELF	CLASS OF SELF	FACTORY OF SELF
		ONLY*1		ONLY*1				
Predefined object identifier NULL	O	O	O	O	O	O	O	O
Predefined object identifier SELF	O *3	X	O *4	X	O	X	O	X
class-name	O *5	X	X	X	O	X	X	X
class-name ONLY	O *5	O *2	X	X	O	X	X	X
FACTORY class-name	X	X	O *5	X	O	X	X	X
FACTORY class-name ONLY	X	X	O *5	O *2	O	X	X	X
UNIVERSAL	X	X	X	X	O	X	X	X
SELF	O *4	X	X	X	O	O	-	-
CLASS OF SELF	O *3	X	X	X	O	-	O	X
FACTORY OF SELF	X	X	O *4	X	O	-	X	O
class-name to be used as an identifier	X	X	O *5	O *2	O	X	X	X

- Items listed under "Sending area" and "Receiving area" represent object reference data items defined by the USAGE clause.
- The predefined object identifiers SELF and NULL or a class-name used as an identifier cannot be used as a receiving item.

O : Indicates that the assignment is possible.

X : Indicates that the description is syntactically possible but not allowed as a rule.

- : Indicates that such a combination is not syntactically allowed.

*1: The left side column shows conformance when ONLY is specified, while the right side is not specified.

*2: The class-name of a sending item must be the same as that of a receiving item.

*3: The statement must appear in the object method of the class inheriting the class-name.

*4: The statement must appear in the factory method of the class inheriting the class-name.

*5: The class specified for the sending object must inherit from the class specified by the receiving object.

Table 11.4 Conformance for Assignment (Object Property and In-Line Method Invocation)

Sending area (Returning item of a method)	Receiving area							
	class-name		FACTORY class-name		UNIVERSAL	SELF	CLASS OF SELF	FACTORY OF SELF
		ONLY*1		ONLY*1				
class-name	O *2	X	X	X	O	X	X	X
class-name ONLY	X	O *2	X	X	O	X	X	X
FACTORY class-name	X	X	O *2	X	O	X	X	X
FACTORY class-name ONLY	X	X	X	O *2	O	X	X	X
UNIVERSAL	X	X	X	X	O	X	X	X
SELF	O *3	O *3	X	X	O	O *3	O *3	X
CLASS OF SELF	O *3	O *3	X	X	O	O *3	O *3	X
FACTORY OF SELF	X	X	O *3	O *3	O	X	X	O *3

- Items listed under "Sending area" and "Receiving area" represent object reference data items defined by the USAGE clause.
- The predefined object identifiers SELF and NULL or a class-name used as an identifier cannot be used as a receiving item.

O : Indicates that the assignment is possible.

X : Indicates that the description is syntactically possible but not allowed as a rule.

*1: The left side column shows conformance when ONLY is not specified, while the right side specifies ONLY.

*2: The class-name of a sending item must be the same as that of a receiving item.

*3: It is assumed that the USAGE OBJECT REFERENCE clause shown in Table 11.6 "Conformance for RETURNING SELF" is used for the sending item. This must be an acceptable combination based on Table 11.4 "Conformance for Assignment (Object Property & In-Line Invocation of a Method.)

Table 11.5 for Assignment (In Case of Special Class)

Sending area	Receiving area			
	Special-class-name (Interface specification no existence)	Special-class-name (Interface specification existence)	Special-class-name to be handled as an identifier	Others
Predefined object identifier NULL	O	O	O	(*1)
Special-class-name (Interface specification no existence)	O (*2)	X	X	X
Special-class-name	O (*3)	O (*2)	X	X

Sending area	Receiving area			
	Special-class-name (Interface specification no existence)	Special-class-name (Interface specification existence)	Special-class-name to be handled as an identifier	Others
(Interface specification existence)				
Special-class-name to be handled as an identifier	X	X	X	X
Others	X	X	X	(*1)

- Items listed under "Sending area" and "Receiving area" represent object reference data items defined by the USAGE clause.

O : Indicates that the assignment is possible.

X : Indicates that the description is syntactically possible but not allowed as a rule.

*1 : Refer to Table 11.3 "Conformance for Assignment (except for Object Property, In-Line Invocation of a Method, and Special Class)."

*2 : For the class specified for a sending area's object and the class specified for a receiving area's class, the literal specified in the class-specifier in the REPOSITORY paragraph must be the same. However, the following combinations are handled as conforming:

- "*OLE" and "*COM"
- "*OLE-ARRAY" and "*COM-ARRAY"
- "*OLE-EXCEPTION" and "*COM-EXCEPTION"

*3 : The class specified in the object on the receiving side must be *COM or *OLE.

11.8.5.5 Conformance of Parameters

Except when the method invoking parameters are omitted, the number of method invoking parameters must match the number of called formal parameters. When either calling parameters or formal parameters are group items, the conformance rules for group items are used. When both parameters are elementary items, the conformance rules for elementary items are used.

11.8.5.5.1 Conformance of Group Items

When either a method invoking parameter or called formal parameter is a group item, the following rules apply:

1. When a parameter or a formal parameter is a group item and neither are strongly typed items, the item corresponding to the group item must be a group item or alphanumeric data item. The size of the formal parameter must be the same as that of the corresponding parameter.
2. When either a parameter or formal parameter is a strongly typed group item, the item corresponding to the group item must also be a group item that is strongly typed with the same type.

11.8.5.5.2 Conformance of Elementary Items to be passed BY CONTENT

If a temporary parameter is a data item with the USAGE OBJECT REFERENCE clause specified, the rules applicable to assignment having arguments on the sending area and formal parameters on the receiving area will apply.

If a formal parameter is a data item with no USAGE OBJECT REFERENCE clause specified, the following rules will apply.

a. If the item of the formal parameter is numeric, the rules applicable to a COMPUTE statement having arguments on the sending area and formal parameters on the receiving area will apply.

b. If the item of the formal parameter is an index, the rules applicable to a SET statement having arguments on the sending area and formal parameters on the receiving area will apply.

c. In cases other than the above, the rules applicable to a MOVE statement having arguments on the sending area and formal parameters on the receiving area will apply.

11.8.5.5.3 Conformance of Elementary Items to be passed by BY REFERENCE

If a formal parameter is a data item with the USAGE OBJECT REFERENCE clause specified, its corresponding argument must comply with the following rules:

a. If the USAGE OBJECT REFERENCE clause with no other optional phrase is specified for the formal parameter, the same clause with no options must be specified for its corresponding argument.

b. If the USAGE OBJECT REFERENCE clause with a class-name is specified for the formal parameter, the same clause with the same class must be specified for its corresponding argument. In addition, if the formal parameter uses the FACTORY and/or ONLY phrase, its corresponding argument must also.

If a formal parameter is a data item with no USAGE OBJECT REFERENCE clause specified, the definitions of the formal parameter and the argument should have the same PICTURE, USAGE, SIGN, SYNCHRONIZED, JUSTIFIES, BLANK WHEN ZERO, and ENCODING clauses, except that:

11.8.5.5.4 Conformance when The ANY LENGTH Clause Is Specified as the Receiving Item

When the ANY LENGTH clause is specified in the description of the pseudo-parameter, the length of the character string of the PICTURE clause of the pseudo-parameter agrees with the length of the character string in the PICTURE clause of the corresponding argument.

11.8.5.6 Conformance of Returning Items

This section describes conformance of returning items when invoking a method.

The returning item specified by the invoked method becomes a sending item, while the returning item specified by the invoking program becomes a receiving item.

11.8.5.6.1 Conformance of Group Items

When either a sending item or receiving item is a group item, the following rules apply:

1. When either a sending or receiving item is a group item and neither are strongly typed items, the item corresponding to the group item must be a group item or alphanumeric data item. The size of the receiving item must be the same as that of the sending item.

2. When either a sending or receiving item is a strongly typed group item, the item corresponding to the group item must also be a group item that is strongly typed with the same type.

11.8.5.6.2 Conformance of Elementary Items

When a sending item is a data item with the USAGE OBJECT REFERENCE clause specified, the following rules apply:

1. When the USAGE OBJECT REFERENCE clause with no SELF phrase specified is a sending item, the sending item must conform to the receiving item. However, the statements of the USAGE OBJECT REFERENCE clauses must match between the sending and receiving items if:

   - The calling item is the INVOKE statement, and an object reference identifier for which the USAGE OBJECT REFERENCE clause with no other select specification is specified is specified for the object to be invoked.

2. When a data item declared with USAGE OBJECT REFERENCE SELF, USAGE OBJECT REFERENCE CLASS OF SELF, or USAGE OBJECT REFERENCE FACTORY OF SELF is specified as for the sending item, it is assumed that the USAGE OBJECT REFERENCE clause listed in Table 11.6, "Conformance for RETURNING SELF," is specified as the sending item and the corresponding assignment rules apply.

   Table 11.6 "Conformance for RETURNING SELF" was created based on the following example:

class definition

```
*> CLASS-ID. class-name.
*> :
 METHOD-ID.  aMethod.
 DATA DIVISION.
  LINKAGE SECTION.
  01 aResult OBJECT REFERENCE SELF.
 PROCEDURE DIVISION  RETURNING aResult.
 END METHOD  aMethod.
*> :
```

```
  01 anObject OBJECT REFERENCE FACTORY OF class-name.
  01 aReturnItem OBJECT REFERENCE class-name-2.
 PROCEDURE DIVISION.
*>   :
     INVOKE anObject "aMethod" RETURNING aReturnItem
```

The columns under "Declarative" show the format for USAGE OBJECT REFERENCE SELF used to define a returning item in the LINKAGE SECTION of a method or method prototype.

The lines under "Object" show the format for USAGE OBJECT REFERENCE used to define an identifier for identifying an invoked object.

To see if this INVOKE statement is valid or not, check Table 11.6 "Conformance for RETURNING SELF" in the Target's 5th line "FACTORY OF class-name" and Declarative's first column for the declarative of a Result, that is, SELF. The result of method invocation should be a class-name (column #1 & line #5). In other words, if class-name matches class-name-2, the INVOKE statement is valid.

Table 11.6 Conformance for RETURNING SELF

Object	Declarative		
	SELF	CLASS OF SELF	FACTORY OF SELF
class-name	-	Class-name	FACTORY OF class-name
class-name ONLY	-	Class-name ONLY	FACTORY OF class-name ONLY
SELF	-	SELF	predefined object identifier SELF *1
CLASS OF SELF	-	CLASS OF SELF	FACTORY OF SELF
FACTORY class-name	Class-name	-	-
FACTORY class-name ONLY	Class-name ONLY	-	-
FACTORY OF SELF	CLASS OF SELF	-	-
predefined object identifier SELF or SUPER	SELF *2	CLASS OF SELF *3	FACTORY OF SELF *3
UNIVERSAL	UNIVERSAL	UNIVERSAL	UNIVERSAL
class-name to be used as an identifier	Class-name ONLY	-	-

- : indicates that this combination is invalid.

*1 : Return values are discarded and replaced by the predefined object identifier SELF.

*2 : Only factory methods are valid.

*3 : Only object methods are valid.

11.8.6 Statements

This section describes the statements native to the object-oriented programming function. Refer to the section "Statements" in Chapter 6, the "Procedure Division" for other statements.

11.8.6.1 The ALTER Statement (Object-Oriented Programming)

Refer to the topic "ALTER Statement " of the "Statements" section of Chapter 6, the "Procedure Division" for rules not native to the object-oriented programming function.

Syntax Rules

The ALTER statement can only be specified in the PROCEDURE DIVISION of a program definition.

11.8.6.2 The CALL Statement (Object-Oriented Programming)

Refer to the topic "CALL Statement (Inter-program Communication)" of the "Statements" section of Chapter 6, the "Procedure Division" for rules not native to the object-oriented programming function.

General Rules

When an object reference identifier is specified as the parameter of the USING or RETURNING phrase in a CALL statement, the USAGE clause of the object reference data item must match the corresponding object reference data item in the called program. Otherwise, results are undefined.

11.8.6.3 The ENTRY Statement (Object-Oriented Programming)

Refer to the topic "ENTRY Statement (Inter-program Communication)" of the "Statements" section of Chapter 6, the "Procedure Division" for rules not native to the object-oriented programming function.

Syntax Rules

The ENTRY statement can only be specified in the PROCEDURE DIVISION of a program definition.

11.8.6.4 The EVALUATE Statement (Object-Oriented Programming)

Refer to the topic "EVALUATE Statement " of the "Statements" section of Chapter 6, the "Procedure Division" for rules not native to the object-oriented programming function.

Syntax Rules

The class of operands before or after the THROUGH phrase should not be an object.

11.8.6.5 The EXIT Statement (Object-Oriented Programming)

The EXIT PROGRAM statement denotes the logical end of a called program. The EXIT METHOD statement denotes the logical end of an invoked method.

Format 1

(For Programs)

EXIT PROGRAM

[RAISING identifier-1]

Format 2

(For Methods)

EXIT METHOD

[RAISING identifier-1]

Syntax Rules

- Rules Common to All Formats

  1. The EXIT statement should not be used in declarative procedures associated with the USE statement with the GLOBAL phrase specified.
  2. The identifier-1 must be an object reference identifier that references the object of the class specified in the RAISING phrase of the PROCEDURE DIVISION header of the source element including the EXIT statement. However when using [Win32] and [Winx64], identifier-1 must not be an object reference identifier of a special class other than *COM-EXCEPTION (or *OLE-EXCEPTION).

- Rules for Format 1

  The EXIT PROGRAM statement can only be specified in the PROCEDURE DIVISION of a program definition.

- Rules for Format 2

  The EXIT METHOD statement can be coded only in the PROCEDURE DIVISION of a method definition.

General Rules

- Rules Common for All Formats

  1. The EXIT statement will fail if it is executed in the scope of a declarative procedure where the GLOBAL phrase is specified in the USE statement, and that USE statement is used in a program or method with the EXIT statement in the format for a method or program.
  2. If the RAISING phrase is specified, an exception condition occurs in the invoking program or method. Then, execution continues in compliance with the rules of the invoking statement. The exception condition is determined in the following manner:

     - If identifier-1 is specified, the object referenced by the identifier will become an exception object.

- Rules for Format 1

  If the EXIT PROGRAM statement is executed in a program not under the control of the calling program or invoking method, the EXIT PROGRAM statement will be handled as if it were the CONTINUE statement. Even if the RAISING phrase is specified, no exception condition will occur.

- Rules for Format 2

  Execution of the EXIT METHOD statement terminates a method and control is returned to the invoking statement. If a method definition containing this statement has the RETURNING phrase, the value of the data item referenced by the RETURNING phrase will become the result of the method invocation.

11.8.6.6 The GO TO Statement (Object-Oriented Programming)

Refer to the topic "GO TO Statement " of the "Statements" section of Chapter 6, the "Procedure Division" for rules not native to the object-oriented programming function.

Syntax Rules

A GO TO statement with procedure-name-1 omitted can only be specified in the PROCEDURE DIVISION of a program definition.

11.8.6.7 The INITIALIZE Statement (Object-Oriented Programming)

Refer to the topic "INITIALIZE Statement" of the "Statements" section of Chapter 6, the "Procedure Division" for rules not native to the object-oriented programming function.

General Rules

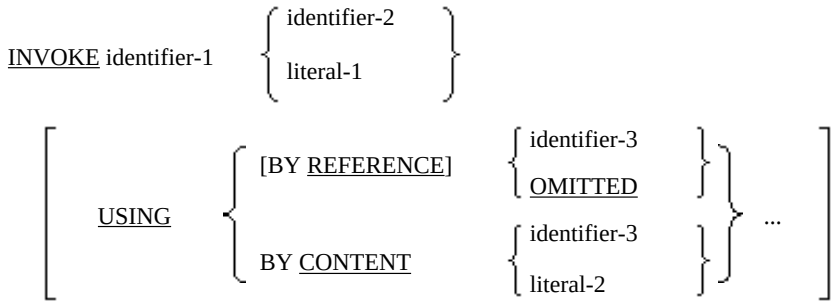
During the process to determine the receiving area's operand for each implicit MOVE, object reference data items are excluded from receiving items.

11.8.6.8 The INVOKE Statement (Object-Oriented Programming)

The INVOKE statement is native to the object-oriented programming function.

The INVOKE statement invokes a method.

Format



[RETURNING identifier-4]

[END-INVOKE]

Syntax Rules

1. Identifier-1 must be an object identifier.
2. Literal-1 must be a nonnumeric literal. If identifier-1 is defined with a class-name, literal-1 must be the name of a method specified by the interface of identifier-1.
3. If identifier-2 is specified, literal-2 should not be specified.
4. The BY CONTENT phrase must not be specified when identifier-1 is one of the following:
   - Object reference identifier for which the USAGE OBJECT REFERENCE clause with no other optional phrase is specified
   - Late-bind special class name
   - Object reference identifier for which a late-bind special class name is specified in the USAGE OBJECT REFERENCE clause
5. Literal-2 must be a value appropriate to the parameter corresponding to the method specified by identifier-1 and literal-1.
6. Identifier-2 must be defined as an alphanumeric data item.
7. Identifier-3 must be defined as the name of a class specified with SELF, NULL, EXCEPTION-OBJECT, or REPOSITORY, or as the FILE SECTION, WORKING-STORAGE SECTION, or LINKAGE SECTION. However, it should not be a class-name of a special class.
8. Identifier-4 must be defined in the FILE SECTION, WORKING-STORAGE SECTION, or LINKAGE SECTION.
9. If BY REFERENCE is used for identifier-3 explicitly or implicitly, it should not be reference-modified.
10. Identifier-4 should not be reference-modified.
11. If identifier-2 is specified, the USAGE OBJECT REFERENCE clause specifying a late-bind special class name or that clause with no other optional phrase must be specified for identifer-1.
12. Identifier-3, literal-2, and identifier-4 must comply with the conformance rules described in the topic "Conformance of Parameters" above.

13. If literal-1 is specified with the USING phrase, the PROCEDURE DIVISION header of the invoked method must contain the USING phrase. If literal-1 is specified with the RETURNING phrase, the PROCEDURE DIVISION header of the invoked method must contain the RETURNING phrase.
14. If the OMITTED phrase is specified, identifier-1 must be a class name or object identifier of a special class.
15. If the BY REFERENCE phrase is specified for identifier-3, EXCEPTION-OBJECT, NULL, SELF, or a class-name cannot be specified as identifier-3.
16. When identifier-3 is a Boolean data item being passed BY REFERENCE (explicitly or implicitly), the internal Boolean data item must be defined to start at a byte boundary.
17. If identifier-4 is an internal Boolean data item, the internal Boolean data item must be defined to start at a byte boundary.
18. When the ANY LENGTH clause is specified as the data item of the LINKAGE SECTION of the method corresponding to identifier-3, identifier-3 must explicitly or implicitly be specified with the BY REFERENCE phrase.
19. When the ANY LENGTH clause is specified in the data item of the LINKAGE SECTION of the method corresponding to literal-2, literal-2 must be either a nonnumeric literal or a national language nonnumeric literal that is not a figurative constant.
20. Indentifier-4 must not be a data item with the ANY LENGTH clause specified.
21. If identifier-1 is a class name or object reference identifier of a special class, literal-2 must not be a figurative constant.
22. If identifier-3 or identifier-4 is a strongly typed group item, the level number of the group item must be 01.

General Rules

1. Identifier-1 identifies an object instance. The content of a data item referenced by literal-1 or identifier-2 identifies the method of the object that executes on that object instance.
    - a. If the method to be invoked is a COBOL method, the content of a data item referenced by literal-1 or identifier-2 must be the external name of the method to be invoked.
2. Literal-2 must not be specified if identifier-1 is one of the following:
    - Object reference identifier for which the USAGE OBJECT REFERENCE clause with no other optional phrase is specified
    - Late-bind special class name
    - Object reference identifier for which a late-bind special class name is specified in the USAGE OBJECT REFERENCE clause
3. Identifier-3 and literal-2 are real parameters for this method invocation.
4. If the RETURNING phrase is specified, the result of the method identified by identifier-1 and identifier-2, or literal-1 will be stored in identifier-4.
5. The rules for binding the INVOKE statement to implementation of a method are provided in the topic "Method Invocation" of the "Compilation Group Structure" section earlier in this chapter.
6. When the INVOKE statement is executed, the specified methods will be enabled for execution and control will return to the invoked method. Control returned from a method moves to the end of the INVOKE statement.
7. If it turns out that the method invoked by the INVOKE statement cannot be executed when the INVOKE statement is executed, the run unit will terminate abnormally.
8. The operation to invoke a method or return from the invoked method is associated with external file connectors and does not change the status or locations of other files.
9. If identifier-2 is specified with the USING phrase, the PROCEDURE DIVISION header of the invoked method must contain the USING phrase. If identifier-2 is specified with the RETURNING phrase, the PROCEDURE DIVISION header of the invoked method must contain the RETURNING phrase.
10. The order of data-names in the USING phrase of the INVOKE statement or the PROCEDURE DIVISION header determines the association of the used data-names between the INVOKE statement and the method to be invoked.

11. If neither the BY CONTENT nor the BY REFERENCE phrase is specified for each argument of the INVOKE statement, the BY REFERENCE phrase is assumed. However, if literal-2 is specified, or if EXCEPTION-OBJECT, NULL, SELF, or a class-name is specified as identifier-3, the BY CONTENT phrase is assumed.
12. The value of a parameter referenced by the USING phrase of the INVOKE statement will become valid when the INVOKE statement is executed, in compliance with the rules described in the topics "Method Invocation" of the "Compilation Group Structure" and "Conformance of Parameters" of the "Procedure Division" section earlier in this chapter.
13. An invoked method can contain an INVOKE statement. An invoked method can execute an INVOKE statement which invokes the invoking method directly or indirectly.
14. The END-INVOKE phrase determines the scope of the INVOKE statement.
15. Identifier-3, literal-2, and identifier-4 must comply with the conformance rules as described in the topic "Conformance of Parameters" of the "Procedure Division" section earlier in this chapter. If, at execution time, the conformance rules are violated, the run unit will terminate abnormally.
16. Control will shift to the invoked method in the manner consistent with the method invocation rules.
17. If the OMITTED phrase is specified, the corresponding argument is regarded as omitted.
18. Refer to the general rules described in section "Procedure Division Header" of Chapter 6, "Procedure Division" for other rules and details of this statement

11.8.6.9 The MERGE Statement (Object-Oriented Programming)

Refer to the topic "MERGE statement (sort-merge)" for details of the rules for the MERGE statement having other than the object-oriented programming function.

General Rules

- Rule for the OUTPUT PROCEDURE phrase

  The EXIT METHOD statement specified in the same source element as that of the MERGE statement cannot be executed in the range of output procedure.

11.8.6.10 The OPEN Statement (Object-Oriented Programming)

Refer to the topic "OPEN Statement (Sequential, Relative, and Indexed Files)" of the "Statements" section of Chapter 6, the "Procedure Division" for rules not native to the object-oriented programming function.

Syntax Rules

The RESERVED phrase of the OPEN statement can only be specified in the PROCEDURE DIVISION of a program definition.

11.8.6.11 The RAISE Statement (Object-Oriented Programming)

The RAISE statement is native to the object-oriented programming function.

This section describes the RAISE statement.

Format

RAISE identifier-1

Syntax Rules

Identifier-1 must be an object reference identifier. However, it must not be an object reference identifier of a special class other than *COM-EXCEPTION (or *OLE-EXCEPTION).

General Rules

1. When the RAISE statement is executed, identifier-1 produces the specified exception object.
2. If there is no applicable declarative procedure, even though identifier-1 has been specified, the RAISE statement will have the same effect as the CONTINUE statement.

11.8.6.12 The SET Statement (Object-Oriented Programming)

The SET statement provides the ability to assign object references.

Format

SET {identifier-1}... TO identifier-2

Syntax Rules

1. Identifier-1 must be used as an object reference
2. Identifier-2 must be a class-name, predefined object identifier, or data item described with USAGE OBJECT REFERENCE.
3. The interface declared for identifier-2 must conform to the interface declared for identifier-1.
4. If identifier-2 is an object reference of a special class, identifier-1 must be an object reference identifier of special class.

General Rules

1. The SET statement retrieves the object reference of the object identified by identifier-2 and stores it in the storage area assigned to identifier-1 of each in the specified order.
2. If identifier-1 and identifier-2 are located in the same memory area, the result of the SET statement is undefined.

11.8.6.13 The SORT Statement (Object-Oriented Programming)

Refer to the topic "SORT statement (sort-merge)" for details of the rules for the SORT statement having other than the object-oriented programming function.

General Rules

- Rule for the INPUT PROCEDURE phrase

  The EXIT METHOD statement specified in the same source element as that of the SORT statement cannot be executed in the range of input procedure.

- Rule for the OUTPUT PROCEDURE phrase

  The EXIT METHOD statement specified in the same source element as that of the SORT statement cannot be executed in the range of output procedure.

11.8.6.14 The STOP Statement (Object-Oriented Programming)

Refer to the topic "STOP Statement" of the "Statements" section of Chapter 6, the "Procedure Division" for rules not native to the object-oriented programming function.

Syntax Rules

Literal-1 of the STOP statement can be coded only in the PROCEDURE DIVISION of a program definition.

11.8.6.15 The USE Statement (Object-Oriented Programming)

This section describes the USE statement.

Format

(Exception Object)

USE AFTER EXCEPTION class-name-1

Syntax Rules

1. Class-name-1 must be the name of a class specified in the REPOSITORY paragraph.
2. When a class-name of a special class is specified as class-name-1, the literal that is specified as the class-specifier of the REPOSITORY paragraph must be the "*COM-EXCEPTION" (or "*OLE-EXCEPTION").

General Rules

1. The runtime system interprets the USE statements in the order they are specified in a program, and declaratives to be executed are selected.
   - If class-name-1 is specified and the exception that has occurred is an object of class-name-1 or a class that inherits from class-name-1, the related declarative will be executed.
2. During execution of related declaratives, the predefined object identifier EXCEPTION-OBJECT references the exception object.

11.8.6.16 The USE FOR DEAD-LOCK Statement (Object-Oriented Programming)

Refer to the topic "USE FOR DEAD-LOCK Statement (Display File)" of the "Statements" section of Chapter 6, the "Procedure Division" for rules not native to the object-oriented programming function.

Syntax Rules

The USE FOR DEAD-LOCK statement can only be specified in the PROCEDURE DIVISION of a program definition.

11.8.6.17 The WRITE Statement (Object-Oriented Programming)

Refer to the topic "WRITE Statement (Sequential File)" of the "Statements" section of Chapter 6, the "Procedure Division" for rules not native to the object-oriented programming function.

Syntax Rules

In the PROCEDURE DIVISION of a method definition not included in a class definition, if you want to use the ADVANCING phrase in the WRITE statement to a file declared in a factory or object definition, one of the following conditions must be satisfied:

- PRINTER or PRINTER-n, where n is an integer from 1 through 9, is specified in the ASSIGN clause for the file.
- The WRITE statement with the ADVANCING phrase for that file is specified in a source element included in the source unit defining that file.
- That file is a print file with the FORMAT clause specified.

11.9 Database (SQL)

Rules to use the database (SQL) function in the class definition are described here.

The function is specific to [Win32], [Winx64] and [Linux].

Refer also to Chapter 8,, "Database (SQL)" for more information.

11.9.1 Reference Format of Embedded SQL

Embedded SQL can be specified in a method definition contained in an object definition or class.

11.9.2 DATA DIVISION

11.9.2.1 Embedded SQL Declarative Section

General Rules

The embedded SQL declarative section can be defined in the WORKING-STORAGE SECTION and LINKAGE SECTION of the DATA DIVISION of the object definition.

11.9.3 Embedded Exception Declarative

General Rules

1. The embedded exception declarative is valid for all SQL statements following the next statement in the source program until an embedded exception declarative that specifies the same condition appears, or until the end of the method is reached.
2. The scope of the valid area of the embedded exception declarative is in units of method.

11.9.4 Cursor System Data Operation Statement

11.9.4.1 Cursor Declarative

Syntax Rules

The cursor-name must be an identifier in the object definition.

General Rules

When two or more objects exist in the same class, each object has its own cursor. When one object is discarded, and when the cursor that is declared in the object is not closed, these cursors are closed by the system.

11.9.5 Dynamic SQL

11.9.5.1 EXECUTE Statement

General Rules

1. The prepared statement that corresponds to the SQL statement identifier must be the PREPARE statement for the SQL statement identifier with the same name within the same object definition, and it must have completed its preparation of the run.
2. When two or more objects exist in the same class, each object has its own SQL statement identifier. When one object is discarded, and when the SQL statement identifier that is declared in the object is in the ready state, these SQL statement identifiers are set to the not-ready state by the runtime system.

11.9.5.2 Dynamic Cursor Declarative

Syntax Rules

1. The cursor-name must be an identifier in the object definition.
2. The SQL statement identifier specified by the dynamic cursor declarative must be specified by the PREPARE statement within the same object definition.

General Rules

When two or more objects exist in the same class, each object has its own cursor and SQL statement identifier. When one object is discarded, and when the cursor that is declared in the object is not closed, these cursors are closed by the runtime system.

When the SQL statement identifier that is declared in the object is in the ready state, these SQL statement identifiers are set in the not-ready state by the runtime system.

11.10 COBOL System Class

COBOL provides the system class to support capabilities required by object-oriented programming.

11.10.1 FJBASE Class

The FJBASE class is the base class; user-defined classes inherit from it. This class provides capabilities required to create and manage objects.

Definition

```
CLASS-ID. FJBASE.
FACTORY.
PROCEDURE DIVISION.
METHOD-ID. CREATE.
END METHOD CREATE.
METHOD-ID. NEW.
END METHOD NEW.
END FACTORY.
OBJECT.
PROCEDURE DIVISION.
METHOD-ID. GETCLASS.
END METHOD GETCLASS.
METHOD-ID. INIT.
END METHOD INIT.
METHOD-ID. _FINALIZE.
END METHOD _FINALIZE.
END OBJECT.
END CLASS FJBASE.
```

11.10.1.1 The CREATE Method

The CREATE method allocates the area for an object and is a factory method to initialize object data to the values specified by the VALUE clause.

Definition

```
METHOD-ID. CREATE.
DATA DIVISION.
 LINKAGE SECTION.
 01 CREATED-OBJECT OBJECT REFERENCE SELF.
PROCEDURE DIVISION RETURNING CREATED-OBJECT.
   *> ...
   EXIT METHOD.
END METHOD CREATE.
```

11.10.1.2 The NEW Method

The NEW method is a factory method used to create an object of a class.

Definition

```
METHOD-ID. NEW.
DATA DIVISION.
 LINKAGE SECTION.
 01 CREATED-OBJECT OBJECT REFERENCE SELF.
PROCEDURE DIVISION RETURNING CREATED-OBJECT.
    INVOKE SELF "CREATE" RETURNING CREATED-OBJECT
    INVOKE CREATED-OBJECT "INIT"
    EXIT METHOD.
END METHOD NEW.
```

11.10.1.3 The GETCLASS Method

The GETCLASS method is an object method which returns an object reference to a factory object of the object's class.

Definition

```
METHOD-ID. GETCLASS.
DATA DIVISION.
 LINKAGE SECTION.
 01 THE-OBJECT-CLASS OBJECT REFERENCE FACTORY SELF.
PROCEDURE DIVISION RETURNING THE-OBJECT-CLASS.
   *> ...
   EXIT METHOD.
END METHOD GETCLASS.
```

11.10.1.4 The INIT Method

The INIT method can be used for initialization operations of objects that could not be performed by the VALUE clause. The FJBASE class provides the INIT method, which does not take any action.

Definition

```
METHOD-ID. INIT.
 PROCEDURE DIVISION.
     EXIT METHOD.
 END METHOD INIT.
```

11.10.1.5 The _FINALIZE Method

The _FINALIZE Method is called automatically when the COBOL system deletes an object.

And, the FINALIZE Method is used during the termination processing of an object.

The _FINALIZE method of the FJBASE class performs no processing.

Definition

```
METHOD-ID. _FINALIZE.
 PROCEDURE DIVISION.
     EXIT METHOD.
 END METHOD _FINALIZE.
```

11.10.2 NULL Classes and NULL Objects

A NULL class is a predefined class. A NULL class has no instance. A NULL object is a NULL factory object. References to NULL objects can be stored in any data item declared with the USAGE OBJECT REFERENCE phrase.

If a method is invoked on a NULL object, the run unit will terminate abnormally.

F

Chapter 12 Microsoft .NET Support

This chapter describes the syntax and semantics for the functions that have been added to NetCOBOL to support operation within the Microsoft .NET (pronounced "dot net") Framework (which we will generally abbreviate as ".NET").

The .NET Framework is a platform that supports the integration of applications and Web services. In particular it provides a common language run-time that supports, in a machine-independent manner, objects compiled from many different programming languages.

In order to support the features of this common environment COBOL needed to be extended. Many concepts of Object-Oriented COBOL apply to the .NET environment. However, there are some differences that have required some new constructs and some Object-Oriented COBOL constructs to be used in a slightly different manner. If you are already familiar with Object-Oriented COBOL you will have a head start when it comes to using NetCOBOL for .NET, but you must check the material in this chapter to be sure you are using the functions in the correct manner.

12.1 Overview

The .NET Framework brings new terminology and functionality to COBOL. Words like "statics", "delegates" and "fields" may be new or different to the COBOL programmer. The reference material in this chapter should therefore be read in conjunction with the online User's Guide and .NET documentation.

The additions to NetCOBOL to support .NET provide the capabilities to:

- Define classes made of static definitions and object definitions
- Define the methods of object instances and statics
- Define interfaces
- Define data (properties) encapsulated in object instances and statics
- Use data types that map to .NET data types
- Inherit from existing classes and define new classes
- Support methods having more than one interface
- Use polymorphism
- Define data items to store references to objects
- Invoke object methods
- Invoke a given method name with different numbers of parameters with different attributes
- Ability to reference fields.
- Create object instances
- Define, instantiate and invoke delegates
- Define and use enumerations
- Invoke unmanaged, or native, (i.e. non-.NET) executables
- Use parameterized types and methods, otherwise known as generics.

12.2 Terminology

The .NET support uses a number of terms defined in the "Terminology" section of "Chapter 11. Object-Oriented Programming Functions". In addition you are likely to encounter the following terms when programming with .NET.

actual parameter/argument

Used to describe the parameter passed in a CALL or INVOKE statement (as opposed to the "formal parameter", or dummy parameter, used to receive the parameter in the called or invoked program/code).

attribute class

A class derived from the .NET System.Attribute class.

class

A group of related variables and methods which describes the behavior of an object in object-oriented programming. A class can be generic, with placeholder type parameters for the types that the class uses, or it can be non-generic, with actual types instead of placeholders. Generics can be used, but not defined, in COBOL.

CLI

Common Language Infrastructure (see below).

CLR

Common Language Runtime (see below).

CLS

Common Language Specification (see below).

CLR type data

Data of a type defined within the CLS. When we talk about a COBOL CLR type data item we mean a COBOL item whose description maps directly to one of the CLR data types. (See Table 12.3)

COBOL-specific type data

Data with a type not defined within the CLS.

collection

A group of the same kind of objects. For example, the format of the collection could be an array or a list.

Common Language Infrastructure (CLI)

The CLI provides a specification for executable code and the execution environment

Common Language Runtime (CLR)

.NET common execution environment that provides runtime support for all languages.

Common Language Specification (CLS)

The CLS is a set of rules intended to promote language interoperability. It is a general, open specification that is supported by .NET. In order to execute in the CLI a language must conform to the CLS.

constraint

A limit that is placed on a type parameter in a generic type or method definition. For example, type parameters can be constrained to types that are reference types or value types, that have a specific base class, that have a default constructor, that implement a specific generic interface, etc. Type arguments in generic types can only be substituted for type parameters if they satisfy the constraints.

custom-attribute

Custom attributes are a general way of extending metadata. They allow you to attach arbitrary additional information to entities in the code, such as classes and methods. Some custom attributes are known to the runtime to have a special purpose. In general, however, there must be an agreement between the person writing the custom attribute and the consumer of the custom attribute for them to have any specific meaning.

delegate

Special purpose reference type that provides a type safe pointer to methods. If you have ever used procedure pointers - pointers that point to procedure code - then delegates provide a similar ability but in a safer manner.

DLLIMPORT attribute

A custom attribute associated with the DllImportAttribute class provided by the .NET system. It is placed on a method signature to indicate that the method is actually defined elsewhere in unmanaged code.

enumeration

An enumeration is a list or collection of like items. Enumerations are quite common in the .NET environment. "Enumeration" is often abbreviated to "enum".

formal parameter

Describes the dummy parameter used to receive the "actual arguments" (parameters) passed from the invoking program or method.

generics

Generics are classes, structures, interfaces, and methods that contain placeholders for the types they use. Using generics, the same source code can be written to apply to many different types in a way that can be verified to be type safe at compile time. This is accomplished by parameterizing source code definitions with type parameters that can represent one of many different types. In addition to offering reuse of code with type safety, generics also allow the runtime environment to generate more efficient code.

generic class

A class that has type parameters for the types that the class uses or stores. Generic class type parameters can be used as placeholders for the types of member variables, method parameters and return values, parameters of other generic types, and of local variables. Generic classes can be used but not created in COBOL.

generic delegate

A delegate with one or more type parameters.

generic interface

An interface that has placeholder type parameters for the types that the interface uses. Generic interface type parameters can be used as placeholders for types used in the interface's method signatures.

generic method

A method that has type parameters for the types that the method uses. Generic method type parameters can be used as placeholders for types used in the method signature as well as for the types of local variables. Generic methods can be associated with both generic and non-generic types. Generic methods always have their own list of type parameters that is distinct from type parameters defined for the enclosing type, if any.

Methods defined within a generic type definition may use the type parameters defined for the generic type. For this reason, such methods are occasionally broadly referred to as generic methods. Methods that have their own type parameters are sometimes referred to as "methods with type parameters" in order to distinguish them.

generic type

A general term to refer to types that have type parameters. A generic type may be open or closed, depending on whether all of its type parameters have been bound to actual types. Generic types are instantiated from generic type definitions when type arguments are substituted for their corresponding type parameters.

[instance] constructor

An instance constructor, generally referred to as a "constructor", initializes object instances.

The method name "NEW" (must be in upper case) of the object definition is used as a keyword in the NetCOBOL for .NET compiler to indicate an instance constructor. Declaring or referencing the object method "NEW" in COBOL source is synonymous with declaring or referencing the instance constructor.

instantiation

The specification of a type argument in a type or method with a generic name, in order to make that type or method operable.

interface

Interfaces can be regarded as a type of contract in which any class that inherits the interface agrees to implement all the methods and properties, using the signatures defined in the interface definition.

intrinsic data types

This phrase is used to describe the native COBOL data types that NetCOBOL uses to support the .NET data types directly (as opposed to the "indirect" support of declaring the .NET data types as OBJECT REFERENCEs).

managed code

Code that executes within the CLR is called "managed code". Code that runs outside the CLR is called "unmanaged code".

metadata

Metadata is data that describes your code. When your code is compiled into a portable executable (PE) format, metadata is inserted into one portion of the file while your code is converted to Microsoft intermediate language (MSIL) and inserted into another. Every type and member defined and referenced in a file or assembly is described within metadata. The metadata and MSIL are contained within EXE or DLL files.

method

A method provides a way to access the data stored in an object or a class. It is a section of code that is associated exclusively with either a class (static method) or an object (instance method). A method consists of statements that perform an action using associated parameters, sometimes with a return value.

method signature

A method signature is the combination of the method's name, the number and types of parameters it receives and the type of any return item.

namespace

Namespaces are a way that classes can be subdivided into logical categories and gives you a way to create globally unique names. For example, the .NET Framework has a top-level namespace called "System", and the NetCOBOL runtime uses "Fujitsu.COBOL". Your applications will use your own (different) namespaces.

open/closed (generic) types

Describes whether a generic type has all type parameters substituted with type arguments. Closed types have all type parameters substituted with type arguments that are themselves non-generic types or closed types. Open types may still have some type parameters that have not been substituted.

overloading/overloaded

.NET supports defining several methods using the same name, distinguished only by their signatures. This feature is called "overloading" and the method names that have multiple methods are said to be "overloaded".

program prototype

A program's profile defined by its name, the form of its parameters and return item.

program prototype definition

A unit of COBOL source that defines the program prototype. Within .NET unmanaged code is defined to the system by using its prototype definition.

reference type

(CLR type) Data that contains the location (reference) of the actual data. Reference types are defined as OBJECT REFERENCEs in COBOL.

static

In the .NET Framework the word "static" is used as a modifier (something which modifies a definition) of methods and data to indicate that they are instantiated only once for the class and not once for each instance. In Net COBOL for .NET static items (methods, data) are declared in the STATIC definition. The STATIC definition is similar to the FACTORY of standard OO COBOL, except that its methods cannot be overridden by methods of subclasses and the methods cannot be invoked dynamically.

Sometimes people will refer to a "static" or "statics", meaning the single instantiation of the code/data in memory. They might also talk about "static objects" but we will avoid this terminology to make a clear distinction between what a STATIC definition creates and what an OBJECT definition creates.

static constructor

The initialization procedure of a static object. In this compiler, the static definition method name "NEW" (all upper-case letters) is treated as a static constructor. Declaring the static method "NEW" in COBOL source is synonymous with declaring a static constructor.

static data

Data that is instantiated only once for a class (i.e. defined in a STATIC definition). All object instances in a class see the same static data values.

static definition

Unit of COBOL source that defines static data and methods.

static method

Method declared in a static definition.

type argument

A type supplied to a generic type or generic method that specifies the actual type to be used in a particular instantiation of a generic type. A type argument is valid as long as it conforms to the constraints defined for the corresponding type parameter.

type object

An object instance of the System. Type class provided by the .NET system.

type parameter

Type parameters are placeholders in generic type definitions and generic method definitions that are replaced by type arguments when a generic type definition or generic method definition is instantiated.

value type

Common Language Runtime (CLR) object. A value type stores data (as opposed to a reference type that stores references to the actual data). When a CLR value type corresponds to a COBOL data type, the compiler makes the appropriate mapping. When a CLR value type does not correspond to a COBOL data type, you need to handle that data as an OBJECT REFERENCE in COBOL.

unmanaged code/program

Code executed within the CLR is called managed code. Code executed outside the CLR, such as Windows APIs, is called unmanaged code.

12.3 Additions to Basic Elements of COBOL

The following topics describe the extensions added to basic elements of the COBOL language to support .NET: COBOL words, maintaining unique references, and defining internal and external names.

12.3.1 COBOL Words

Support for .NET requires extensions to both user-defined words and reserved words as described below.

12.3.1.1 User-Defined Words

.NET support uses the following types of user-defined words which are described in subsequent topics:

- Class-name
- Custom-attribute-name
- Delegate-name
- Enum-name
- Field-name

- Interface-name
- Method-name
- Program-prototype-name
- Property-name

Class-name

A class-name identifies a class that defines common behaviors and attributes to be shared by zero or more objects.

Custom-attribute-name

A custom-attribute-name identifies a custom attribute.

Delegate-name

A delegate name identifies a delegate. A delegate is an object-oriented equivalent of a function pointer. Unlike function pointers, delegates are object-oriented, type-safe and secure.

Enum-name

An enum-name identifies an enumeration of items of a common type.

Field-name

A field name identifies an elementary value or a group item. Fields are defined in the WORKING-STORAGE SECTION of either the STATIC or OBJECT definition.

Interface-name

An interface name identifies an interface, defined by a set of method prototypes.

Method-name

A method-name identifies a method.

Method-identifier

A method-identifier identifies the combination of the method with the type parameter and type specified as the type argument.

Program-prototype-name

A program-prototype-name is the name of the type of the program. It is used to identify called programs.

Property-name

A property-name identifies an object property and is a means of getting information out of and passing information back into an object.

12.3.1.2 Reserved Words

The .NET extensions add the following COBOL reserved words:

- BINARY-CHAR
- BINARY-SHORT
- BINARY-LONG
- BINARY-DOUBLE
- CUSTOM-ATTRIBUTE
- DELEGATE
- DELEGATE-ID
- ENUM
- ENUM-ID
- INTERFACE

- INTERFACE-ID
- INTERNAL
- PRIVATE
- PUBLIC
- STATIC

12.3.2 Unsupported Words

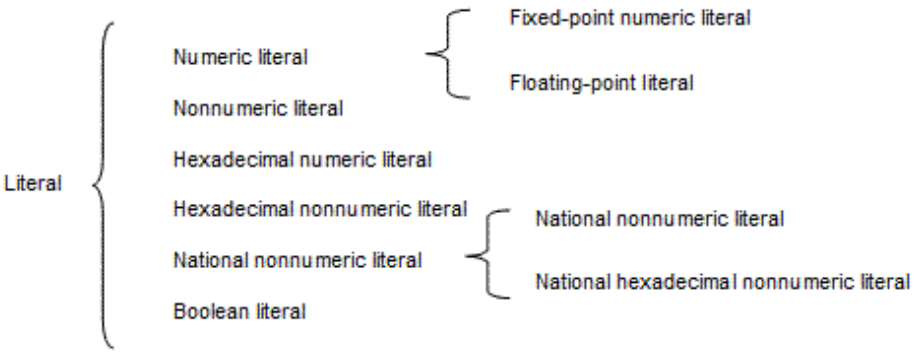
The nature of .NET COBOL prevents the use of certain verbs that may be common in batch or single-processor programs. Below is a list of verbs that may not be used in .NET COBOL:

- INITIATE
- GENERATE
- SUPPRESS
- TERMINATE
- USE AFTER
- ACCEPT (Screen Operation)
- DISPLAY (Screen Operation)

12.3.3 Literal

This section describes the rules of literals specific to NetCOBOL for .NET. For other rules, see the topic titled "Literal" in the "Basic Overview of the Language" section of Chapter 1, "General Rules".

A literal is data having a constant value. There are several literal types as shown below.



12.3.3.1 Numeric Literals

Fixed-point Literal

The total number of digits in the integer part and decimal part must be between 1 and 19.

Hexadecimal Numeric Literal

A hexadecimal numeric literal is a hexadecimal character string whose value represents a numeric value.

Format

H"hexadecimal-character-string-1"

1. The characters in hexadecimal-character-string-1 must be delimited by the separator H" at the left and by double quotation marks at the right.
2. Hexadecimal-character-string-1 must consist of characters between 0 and 9 or A to F.
3. The number of characters in hexadecimal-character-string-1 must be an even number between 2 and 16.
4. Hexadecimal-character-string-1 must be represented in big-endian format.
5. The hexadecimal numeric literal category is numeric.
6. A hexadecimal numeric literal can be specified wherever "literal-n" is indicated in a syntax diagram, and where a numeric literal is permitted.
7. The hexadecimal numeric literal cannot represent a decimal fraction or a negative number.
8. The hexadecimal numeric literal is aligned according to the standard alignment rules.

12.3.4 Uniqueness of Reference

This section describes how uniqueness of reference is preserved in NetCOBOL for .NET.

12.3.4.1 Identifier

- Format 1

  (Function identifier)

  function-identifier-1

- Format 2

  (Qualified data-name with subscript)

  data-name-1 [{ IN / OF } data-name-2] ... [{ IN / OF } file-name-1]

  [{subscript-1}...]

- Format 3

  (Reference-modification)

  identifier-1 reference-modifier-1

- Format 4

  (In-line-method-invocation)

  in-line-invocation-1

- Format 5

  (Object modifier)

  identifier-2 object-modifier-1

- Format 6

  (Predefined object identifier)

  { EXCEPTION-OBJECT / NULL / SELF }

SUPER

- Format 7

  (Object property)

```
property-name-1 OF { identifier-1 }
                   { class-name-1 }
```

- Format 8

  (TYPE OF operator)

```
TYPE OF { class-name-2     }
        { interface-name-1 }
        { delegate-name-1  }
        { enum-name-1      }
```

- Format 9

  (Field reference)

```
field-name-1 OF { identifier-4 }
                { class-name-3 }
                { enum-name-2  }
```

Syntax Rules

- All Formats

  Identifiers are defined recursively. Identifiers can take any format permitted by the rules for the format being considered.

- Format 1 (Function identifier)

  Function-identifier -1 is a function identifier as defined in the "Function Identifier" topic in the "Uniqueness of Reference" section in Chapter 1.

- Format 2 (Qualified data-name with subscript)

  1. The words IN and OF are equivalent.
  2. If data-name-1 is not unique in the scope of names of the current source element, it should be followed by a syntactically correct combination of qualifiers and subscripts required for uniqueness of reference. See topic "Scope of Names" in the "Inter-program Communication Module" section of Chapter 2, "COBOL Modules".
  3. Refer to the topic "Subscripting" within "Uniqueness of Reference" in Chapter 1 for the definition of subscripts.

- Format 3 (Reference modifier)

  Reference-modifier-1 is described in the topic "Reference Modification" in the "Uniqueness of Reference" section of Chapter 1.

- Format 4 (In-line method invocation)

  In-line-invocation-1 is described in topic "In-line Method Invocation"

- Format 5 (Object modifier)

  See the topic "Object Modifiers" for the details of object-modifiers.

- Format 6 (Predefined object identifier)

  See the "Predefined Object Identifiers" topic for the details of predefined object identifiers.

- Format 7 (Object property)

  See the topic "Object Property" for the details of object properties.

- Format 8 (TYPE OF operator)

  See the topic "TYPE OF Operator" for details.

- Format 9 (Field reference)

  See the topic "Field References" for details.

General Rules

The rules for uniqueness of reference apply in the order listed below:

a. Predefined object-identifiers are elementary identifiers.

b. The TYPE OF operator is applied to the class-name, interface-name, delegate-name or enum-name on the right.

c. For data-name qualification, IN or OF qualifies the (possibly already qualified) data-name on the left with a file-name, or data-name on the right.

d. The subscript applies to the fully qualified data-name on the left.

e. The object-modifier applies to the object-identifier on the left.

f. For the object property, OF applies the property-name on the left to the object-identifier on the right.

g. For a field reference, OF applies the field-name on the left to the object-identifier on the right.

h. The invocation operator for in-line method invocation applies a literal method-name with optional parameters enclosed in parentheses on the right to the object-identifier on the left.

i. Without arguments, a function-identifier is assumed to be an elementary identifier. If specified with arguments, the function-name will be applied to the argument list enclosed in parentheses.

j. The reference-modifier applies to the identifier on the left.

Object Identifiers

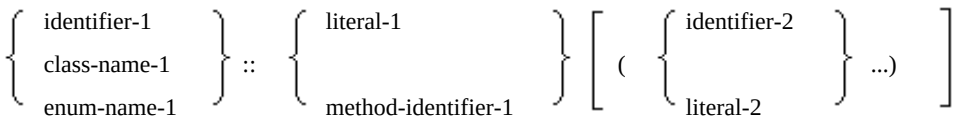
The following are all referred to as object [reference] identifiers:

- References for a data description entry specifying a USAGE OBJECT REFERENCE clause.
- Inline method invocations.
- Object properties.
- Predefined object identifiers.
- Field references.

12.3.4.2 In-line Method Invocation

In-line invocation of a method references the value returned from the invoked method.

Format



Syntax Rules

1. Inline method invocation shall not be specified for a receiving item.
2. Identifier-1 must be an object identifier, but it cannot be NULL.
3. Class-name-1 and enum-name-1 should be names defined in the REPOSITORY paragraph respectively as a class-specifier and enum-specifier.

4. Literal-1 must be a nonnumeric literal, or national language nonnumeric literal, of which the value is the method name specified for identifier-1, class-name-1 or enum-name-1.
5. The method invoked by this format must have a return item.
6. Identifier-2 must be SELF, EXCEPTION-OBJECT, or NULL, or a data item defined in the FILE, WORKING-STORAGE, LINKAGE, or LOCAL-STORAGE section.
7. Literal-2 must be valid as the sending item of a MOVE statement to the item of the same class and category as the corresponding parameter of the invoked method.
8. Identifier-2 must be declared so that corresponding parameters conform.
9. When an internal Boolean data item is specified for identifier-2, the internal Boolean data item must be aligned on a byte boundary.
10. Reference modification should not be used on identifier-2.

General Rules

1. Identifier-1, class-name-1, and enum-name-1 are objects on which a method is invoked.
2. Literal-1 or method-identifier-1 identifies the call method.
3. Using in-line method invocation gives the same behavior as using temporary-data-name after either of the following statements:

```
INVOKE call-object "Method name"
                           USING arguments
                           RETURNING temporary-data-name
INVOKE call-object "Method name"
                           RETURNING temporary-data-name
```

   a. If arguments are given, they are parameters specified for in-line method invocation.
   b. The temporary-data-name has the same description, class, and category as the returning item for the method specified by identifier-1 (or class-name-1, enum-name-1) and literal-1.
   c. The temporary-data-name is a temporary item that exists for in-line invocation only.

12.3.4.3 Object Modifiers

An object-modifier is a syntactically correct combination of character-strings and separators that uniquely reference an object reference identifier through the specified description.

Format

identifier-1 AS { class-name-1 | interface-name-1 | delegate-name-1 | enum-name-1 }

Note

identifier-1 in the above format is used to show the context and is not a part of the object-modifier.

Syntax Rules

1. Identifier-1 must be an object reference identifier. However, it should not be the predefined object identifier SUPER or NULL.
2. Identifier-1 should not be a receiving item.

3. Class-name-1, interface-name-1, delegate-name-1, and enum-name-1 should be names defined in the REPOSITORY paragraph respectively as a class-specifier, an interface-specifier, a delegate-specifier and an enum-specifier.

General Rules

1. The reference of identifier-1 is treated as though it had the description specified by the AS phrase.
2. When class-name-1 is specified, the implicit description of the result is "USAGE OBJECT REFERENCE class-name-1". If the object identified by identifier-1 is neither an object of class-name-1 nor an object of a subclass of class-name-1, an exception is generated.
3. When interface-name-1 is specified, the implicit description of the result is "USAGE OBJECT REFERENCE interface-name-1". If the object identified by identifier-1 does not implement the interface defined by interface-name-1, an exception is generated.
4. When delegate-name-1 is specified, the implicit description of the result is "USAGE OBJECT REFERENCE delegate-name-1". If the object identified by identifier-1 is not an object of delegate-name-1, an exception is generated.
5. When enum-name-1 is specified, the implicit description of the result is "USAGE OBJECT REFERENCE enum-name-1". If the object identified by identifier-1 is not an object of enum-name-1, an exception is generated.

12.3.4.4 Predefined Object Identifiers

The predefined object identifiers are:

- EXCEPTION-OBJECT
- NULL
- SELF
- SUPER

12.3.4.4.1 EXCEPTION-OBJECT

EXCEPTION-OBJECT is a predefined object identifier used in declaratives to reference the current exception object.

Format

EXCEPTION-OBJECT

Syntax Rules

1. EXCEPTION-OBJECT can be used only in a declarative procedure.
2. EXCEPTION-OBJECT cannot be used as a receiving operand.

General Rules

1. EXCEPTION-OBJECT references the current exception object. If an exception object is not associated with the current exception, EXCEPTION-OBJECT is set to null.
2. EXCEPTION-OBJECT is implicitly declared as "USAGE OBJECT REFERENCE class-name" where class-name is the class specified in the USE statement of the declarative procedure containing the EXCEPTION-OBJECT.

12.3.4.4.2 NULL

NULL is a predefined object identifier that contains a null reference.

Format

NULL

Syntax Rules

NULL cannot be specified as a receiving item.

General Rules

NULL or the null object always refers to the same object (i.e. so they can be compared and found equal).

12.3.4.4.3 SELF and SUPER

SELF and SUPER are predefined object identifiers that reference the object on which the current method is executing.

Format

```
{ SELF  }
{ SUPER }
```

Syntax Rules

1. These object identifiers can only be used in the PROCEDURE DIVISION of methods within the OBJECT definition.
2. These object identifiers cannot be used for receiving items.
3. SUPER can be used as:
   - the target in an INVOKE statement,
   - the object in an in-line method invocation, or
   - an object property.

General Rules

1. Predefined object identifiers, SELF or SUPER, reference the object in which SELF or SUPER appears and are used to invoke a method of that object.
2. If SELF is specified to invoke a method, the method resolution is based upon the set of methods defined for the object.
3. If SUPER is specified to invoke a method, the method resolution ignores all the methods defined in the class containing the invocation and all the methods defined in any subclass of that class.

12.3.4.5 Object Property

The object property provides a special syntax to reference the information of an object or to set information for an object.

Format

```
property-name-1 OF { identifier-1 }
                   { class-name-1 }
```

Syntax Rules

1. Identifier-1 must be an object identifier, but it cannot be NULL.
2. When an object property is used as a sending item, a GET property method for property-name-1 must exist in the object referenced by identifier-1 or STATIC definition of class-name-1.
3. When an object property is used as a receiving item, a SET property method for property-name-1 must exist in the object referenced by identifier-1 or STATIC definition of class-name-1.

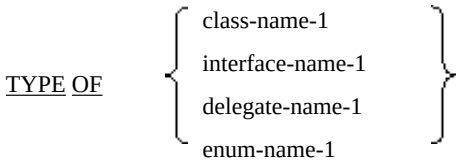
4. An object property to be used as a sending item must be described in the same way as the returning item of the GET property method. This object property can be used in any place where a data item can be used as a sending item.
5. An object property to be used as a receiving item must be described in the same way as the USING parameter of the SET property method. This object property can be used in any place where a data item can be used as a receiving item.
6. The data description entry for the item specified in the RETURNING phrase of the GET property method must be the same as the data description entry for the item specified in the USING phrase of the SET property method.
7. An object property must not be used with reference modification.
8. Object properties, when used as receiving items, can only be specified in:
   - The receiving item of a MOVE statement without a CORRESPONDING phrase
   - The receiving item of a SET statement

General Rules

1. If an object property is only used as a sending item, a conceptual temporary data item, temp-1, is used. The value of the property is determined when the associated get property method is executed and a returned value placed in temp-1. The data description for temp-1 is the same as that specified in the RETURNING phrase of the GET property method.
2. If an object property is only used as a receiving item, a conceptual temporary data item, temp-2, is used. The value of the property is assigned when the associated SET property method is executed and the content of temp-2 is moved to the parameter. The data description for temp-2 is the same as that specified in the USING phrase of the SET property method.

12.3.4.6 TYPE OF Operator

Format



Syntax Rules

1. Class-name-1, interface-name-1, delegate-name-1, and enum-name-1 should be names defined in the REPOSITORY paragraph respectively as a class-specifier, an interface-specifier, a delegate-specifier and an enum-specifier.
2. This format cannot be used for receiving items.

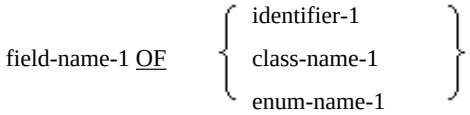
General Rules

The TYPE OF operator acquires the type object (an object of the System.Type class) that describes the object specified in class-name-1, interface-name-1, delegate-name-1 or enum-name-1.

12.3.4.7 Field References

A field reference provides the means of referring to a field.

Format



Syntax Rules

Identifier-1 must be an object reference identifier, but it cannot be SUPER and NULL.

General Rules

1. When identifier-1 is specified, field-name-1 references a field that is not a static attribute.
2. When class-name-1 is specified, field-name-1 references an item that has the static attribute declared explicitly or implicitly in the class.
3. When enum-name-1 is specified, field-name-1 references an element in the enumeration.
4. Fields (in COBOL) can be level 01 or 77 items declared in the WORKING-STORAGE SECTION of a static or object definition. A field declared in a static definition has a static attribute. A field declared in an object definition does not have a static attribute.
5. You cannot reference a field that has variable length.
6. A field reference must not be used with reference modification.
7. If field-name-1 is a literal field, its field reference cannot be used for receiving items.

12.3.5 External and Internal Names

An externalized name is a name stored in metadata, and can be referenced from other programs and classes. You can use names defined outside of the COBOL system by using external names within your COBOL programs.

Internal names, on the other hand, are only accessible from within the separately compiled COBOL program. They are defined according to the rules of creating user-defined words in COBOL.

The following elements have an external name and an internal name:

- Class
- Interface
- Delegate
- Enumeration
- Program (Except internal program)
- Program prototype
- Method
- Property
- Field
- Parameter and Return value

You associate external and internal names using one of the following:

- Specifiers in the REPOSITORY paragraph. See the "Repository Paragraph" section for details of these specifiers. These specifiers allow you to refer to metadata names outside of the current source unit and may require all of the forms of name modification listed below.
- Entries in the IDENTIFICATION DIVISION (for Class, Interface, Delegate, Enumeration, Program, Program Prototype, Method, Property method definition). See "Identification Division" for details of how to define these external names. These external names specify the names that are persisted into metadata based on definitions.
- Using the AS phrase in a data description. See "Accessibility Specifier Phrase" for details. These external names specify the names of method parameters that are persisted into metadata.

When you do not explicitly specify an external name, the system assumes that the external name is the same as the internal name. External names must be unique within each compilation unit.

Unless specified in this chapter that a name is an external name, you can assume that it is an internal name.

12.3.5.1 External Names

12.3.5.1.1 Name modification with namespaces or nesting

An external name contains elements (modifiers) for each higher-level namespace or enclosing class (if nesting is used) of the item being referenced. (Note: defining nested classes is not supported in NetCOBOL for .NET but it is supported in some other .NET programming languages.) External names can be defined for classes, interfaces, delegates, enumerations, programs and program prototypes. The combination of the item name, namespace modifiers and enclosing type modifier produces a fully specified name.

If there are neither namespaces nor an enclosing type, name modification is unnecessary. The rules for name modification with namespaces or nesting are listed below:

a. Namespace modifiers are separated by periods.

b. A nested type is separated from its enclosing type by "+" (the plus character).

c. When an enclosing type has type parameters, the modification shown in "Modification for type parameters" is necessary.

Modification is never required for methods, properties or fields, as the context is determined by the containing class, so, for these items, the characters ".", and "+" are interpreted as being part of the item's name.

When the external name doesn't satisfy the condition a to c listed above, "." and "+" are also interpreted as being part of the item's name.

For example, if "classA" belongs to namespace "FJnamespace" its external name is "FJnamespace.ClassA" (1). If there were an enumeration "enumB" nested within "classA", the enumeration's external name would be "FJnamespace.ClassA+enumB" (2).

```
CLASS CLASS-A  AS "FJnamespace.ClassA"              *> (1)
ENUM  ENUM-B   AS "FJnamespace.ClassA+enumB"        *> (2)
```

12.3.5.1.2 Modification for Array Types

The literal in a class-specifier in the REPOSITORY section can indicate an array type by including brackets ([ ]) after the class name string in the literal.

The rules for array type external names are listed below:

a. You can only enter one set of brackets after the class name, i.e., arrays of arrays (also known as jagged arrays) cannot be specified in NetCOBOL.

b. You can only enter commas (,) in the brackets. The commas indicate the number of dimensions of the array; no commas mean a single dimensional array; one comma means a two dimensional array and so on. Note that all such arrays are zero-based.

c. The closing bracket must be the last character in the literal.

d. If conditions a-c are not satisfied, it is assumed that an array type has not been specified and that any brackets are part of the class name.

e. The type name must be a fully specified class name or a delegate name.

f. If the type has type parameters, the modification shown in "Modification for type parameters" is necessary.

For example, a one-dimensional array of "System.Object" type can be specified in the REPOSITORY as in (1). A three-dimensional array of "System.Int32" type can be specified as in (2).

```
CLASS ARRAY-OBJECT AS "System.Object[]"      *> (1)
CLASS D3ARRAY-INT  AS "System.Int32[,,]"     *> (2)
```

12.3.5.1.3 Modification for Type Parameters

If you specify a generic class, a generic interface, a generic delegate, or a generic method in the REPOSITORY section for a class-specifier, an interface-specifier, a delegate-specifier, or a method-specifier respectively, the literal used in the specifier indicates that it is a generic type or method by including angular brackets (<>) after the name of the type or method.

The rules for indicating the presence of type parameters are listed below:

a. You can enter only one set of angular brackets after the type or method name.

b. You can enter only comma separators (,) in the angular brackets. The number of commas indicates the number of type parameters: no commas means a single type parameter, one comma means two type parameters, etc.

c. Type names must be fully specified, i.e., include namespace and nesting modification.

For example, the System.Collections.Generic.List type that has one type parameter may be specified as in (1). System.Collections.Generic.Dictionary, which has two type parameters, may be specified as in (2).

```
CLASS G-List  AS  "System.Collections.Generic.List<>"         *> (1)
CLASS G-Dict  AS  "System.Collections.Generic.Dictionary<,>"  *> (2)
```

A one dimensional array of System.Collections.Generic.List, which has one type parameter, may be specified as in (3). The generic System.Collections.Generic.Dictionary type has two type parameters and a nested Enumerator type that may be specified as in (4).

```
CLASS G-List-ARY  AS  "System.Collections.Generic.List<>[]"  *> (3)
CLASS G-Dict-Enumerator  AS
    "System.Collections.Generic.Dictionary<,>+Enumerator"    *> (4)
```

12.4 Compilation Group Structure

This section describes how the different parts that form programs for .NET are structured.

12.4.1 Organization

A compilation group can contain one or more units of COBOL source. (Note, nested programs and a sequence of separate programs in a single file are not supported.)

Statements, entries, paragraphs, and sections of a source unit are grouped into the four divisions appearing in the following order, except for source text manipulation statements and end markers:

1. IDENTIFICATION DIVISION
2. ENVIRONMENT DIVISION
3. DATA DIVISION
4. PROCEDURE DIVISION

The beginning of a division in a source unit is indicated by the appropriate division header. The beginning of the IDENTIFICATION DIVISION can be denoted by one of the paragraph headers allowed for the IDENTIFICATION DIVISION.

The end of a division in a source unit is indicated by:

- The beginning of the succeeding division within that source unit
- The end marker of that source unit
- There being no more source lines following the division

The end of a COBOL source unit is indicated explicitly by an end marker, or implied by the fact that no more source lines follow.

A source unit directly or indirectly included in another source unit is assumed to be a different source unit that can refer to, or add itself to, the resources of the source unit including it.

12.4.1.1 COBOL Compilation Group

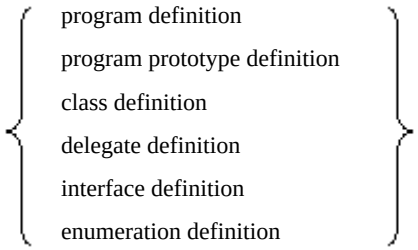
A compilation group is made up of one or more source units, and each source unit is one of: a program definition, program prototype definition, class definition or delegate definition.

The following diagrams define the formats for the parts that make up the compilation group:

Compilation Group

[{ source unit}]

Source Unit



Program Definition Format

[IDENTIFICATION DIVISION .]

PROGRAM-ID. program-name-1 [AS program-name-literal-1]

[Environment division]

[Data division]

[Procedure division]

[{ Program Definition }]

[END PROGRAM { program-name-1 / program-name-literal-1 } .]

Program Prototype Definition Format

[IDENTIFICATION DIVISION .]

PROGRAM-ID. program-prototype-name-1
[AS program-prototype-name-literal-1]
IS PROTOTYPE.

[Environment division]

[Data division]

[Procedure division]

END PROGRAM program-prototype-name-1.

Class Definition Format

[IDENTIFICATION DIVISION .]

CLASS-ID. class-name-1 [AS class-name-literal-1]

[Environment division]

[Static definition]

[Object definition]

END CLASS class-name-1.

Static Definition Format

```
[ IDENTIFICATION DIVISION .]
STATIC.
[ Environment division ]
[ Data division ]
[ PROCEDURE DIVISION.
[{ Method definition }...] ]
END STATIC.
```

Object Definition Format

```
[ IDENTIFICATION DIVISION .]
OBJECT..
[ Environment division ]
[ Data division ]
[ PROCEDURE DIVISION.
[{ Method definition }...] ]
END OBJECT.
```

Method Definition Format

```
[ IDENTIFICATION DIVISION .]
METHOD-ID. method-name-1.
[ Environment division ]
[ Data division ]
[ Procedure division ]
END METHOD [ method-name-1 ].
```

Delegate Definition Format

```
[ IDENTIFICATION DIVISION .]
DELEGATE-ID. delegate-name-1.
[ Environment division ]
[ Data division ]
[ Procedure division ]
END DELEGATE delegate-name-1.
```

Interface Definition Format

```
[ IDENTIFICATION DIVISION .]
INTERFACE-ID. interface-name-1.
[Environment division ]
```

[PROCEDURE DIVISION.
[{Method definition }...]]

END INTERFACE interface-name-1.

Enumeration Definition Format

[IDENTIFICATION DIVISION .]

ENUM-ID. enum-name-1.

[Environment division]

[Data division]

END ENUM enum-name-1.

Syntax Rules

1. PROGRAM-ID, CLASS-ID, OBJECT, METHOD-ID, DELEGATE-ID, STATIC, INTERFACE-ID and ENUM-ID Paragraph Formats and Syntax Rules

   Complete formats and rules for these paragraphs are defined in the section "Identification Division" .

2. Program Definition Description

   A program definition is a unit of source that starts with an identification division containing a PROGRAM-ID paragraph. It can include an ENVIRONMENT DIVISION, a DATA DIVISION and a PROCEDURE DIVISION. A program definition can contain other program definitions. A program definition must end with an END PROGRAM header when:

   a. A program definition contains one or more other program definitions.

   b. A program definition is contained in another program.

3. Program Prototype Definition Description

   A program prototype definition is a program definition with the PROTOTYPE attribute specified in the PROGRAM-ID paragraph.

   The data division of a program prototype definition can only include the LINKAGE SECTION.

   The procedure division should only contain the PROCEDURE DIVISION header.

   A program prototype definition should be terminated with an END PROGRAM marker. Note that you can only use the internal name (program-prototype-name-1) on the END PROGRAM marker.

4. Class Definition Description

   A class definition is a unit of source that starts with an identification division containing a CLASS-ID paragraph. It can include an ENVIRONMENT DIVISION, a static definition, and an object definition.

   Definitions in the environment division apply to the class definition and the other included definitions.

   The environment division must not contain an INPUT-OUTPUT SECTION.

   A class definition should be terminated with an END CLASS marker.

5. Static Definition Description

   A static definition starts with an IDENTIFICATION DIVISION containing a STATIC paragraph. It can include an ENVIRONMENT DIVISION, DATA DIVISION and PROCEDURE DIVISION.

   Definitions within the environment division apply to the static definition and all methods declared in the static definition.

   The environment division must not contain a CONFIGURATION SECTION.

   The data division can only include a BASED-STORAGE SECTION, a FILE SECTION, a WORKING-STORAGE SECTION and a CONSTANT SECTION.

The procedure division can only contain method definitions. It cannot contain other statements or paragraph and section headers.

A static definition should be terminated with an END STATIC marker.

6. Object Definition Description

   An object definition starts with an IDENTIFICATION DIVISION containing an OBJECT paragraph. It can include an ENVIRONMENT DIVISION, DATA DIVISION and PROCEDURE DIVISION.

   Definitions within the environment division apply to the object definition and all methods declared in the object definition.

   The environment division must not contain a CONFIGURATION SECTION.

   The data division can only include a BASED-STORAGE SECTION, a FILE SECTION, a WORKING-STORAGE SECTION and a CONSTANT SECTION.

   The procedure division can only contain method definitions. It cannot contain other statements or paragraph and section headers.

   An object definition should be terminated with the END OBJECT marker.

7. Method Definition Description

   A method definition starts with an IDENTIFICATION DIVISION containing a METHOD-ID paragraph. It can include an ENVIRONMENT DIVISION, DATA DIVISION and PROCEDURE DIVISION.

   The environment division must not contain a CONFIGURATION SECTION.

   The data division can only include a BASED-STORAGE SECTION, a FILE SECTION, WORKING-STORAGE SECTION, a LINKAGE SECTION and a CONSTANT SECTION.

   The procedure division follows the standard rules for procedure divisions.

   A method definition should be terminated with the END METHOD marker.

8. Delegate Definition Description

   A delegate definition starts with an IDENTIFICATION DIVISION containing a DELEGATE-ID paragraph. It can include an ENVIRONMENT DIVISION, DATA DIVISION, and a PROCEDURE DIVISION.

   Definitions within the environment division apply only to the delegate definition.

   The Environment Division can include only a CONFIGURATION SECTION.

   The Data Division can include only a LINKAGE SECTION.

   The Procedure Division should only contain the PROCEDURE DIVISION header.

   A delegate definition should be terminated with an END DELEGATE marker.

9. Interface Definition Description

   An interface definition starts with an IDENTIFICATION DIVISION containing an INTERFACE-ID paragraph. It can include an ENVIRONMENT DIVISION, and a PROCEDURE DIVISION.

   The ENVIRONMENT DIVISION of an interface definition can include only the CONFIGURATION SECTION.

   The PROCEDURE DIVISION of an interface definition can include only a method definition. The PROCEDURE DIVISION of an interface definition cannot include paragraph markers, statements and clauses.

   An interface definition should be terminated with an END INTERFACE marker.

10. Enumeration Definition Description

    An enumeration definition starts with an IDENTIFICATION DIVISION containing an ENUM-ID paragraph. It can include an ENVIRONMENT DIVISION, and DATA DIVISION.

    The ENVIRONMENT DIVISION of an enumeration definition can include only the CONFIGURATION SECTION.

    The DATA DIVISION can include only a WORKING-STORAGE SECTION.

    An enumeration definition should be terminated with an END ENUM marker.

12.4.2 Generics

Generics allow the definition of source code constructs in which the exact type of a particular construct is left unspecified. Such unspecified types are represented using placeholders, called type parameters. NetCOBOL supports the use of generic source code constructs that have been defined in other languages, but does not support their definition. This section describes characteristics of generic types and methods defined in other languages that can be used in NetCOBOL.

When a generic definition is used or instantiated, all previously unspecified types must be replaced with actual types. Types specified in such an instantiation are referred to as type arguments. It is also possible for a generic definition to place limitations or constraints on the actual type that may be substituted during instantiation. The validity of a type argument is checked at compile time against any constraints defined for the corresponding type parameter in the definition of the generic source element.

Generic Types

Generic types are types with one or more type parameters. Any of the following can be generic types:

- Class
- Interface
- Delegate

.NET assemblies with generic types define type parameters that can be used to specify the following:

- The type of a static or instance field.
- The type of a return value or a method parameter.
- The type arguments and constraints of other type parameters.

Since generic types contain elements with an undecided type, they cannot be used like a normal type. However, when appropriate type arguments are supplied for all type parameters, the type becomes a closed generic type and may be used as any other type.

Generic Methods

Generic methods can have one or more type parameters. Type parameters can be used to define methods that specify the following:

- The types of method parameters or the type of the return value.
- Types of local data in a method.
- The type arguments and constraints of other type parameters.

Since generic methods contain elements with an undecided type, they cannot be used like a normal method. However, when appropriate type arguments are supplied for all type parameters, the method that results from the substitution of the type arguments may be used in contexts where any other method might be used.

Type arguments can be specified for generic methods in one of two ways: they may either be deduced from the invocation or they may be specified explicitly using a method specifier in the REPOSITORY section. In order for the type arguments to be deduced from the list of method arguments, as opposed to specified explicitly, the following conditions must be met:

1. The return type may not be a type parameter.
2. None of the actual parameters whose type is a type parameter in the formal parameter list may be a COBOL original type.
3. The type of each type argument can be uniquely identified from the actual parameters.

12.5 Identification Division

The IDENTIFICATION DIVISION is used to identify which unit of source you are describing. The different units are: program definition, program prototype definition, class definition, static definition, object definition, method definition, delegate definition, interface definition and enum definition.

The IDENTIFICATION DIVISION header is optional.

Format

```
[IDENTIFICATION DIVISION.]
    { PROGRAM-ID paragraph
      CLASS-ID paragraph
      STATIC paragraph
      OBJECT paragraph
      METHOD-ID paragraph
      DELEGATE-ID paragraph
      INTERFACE-ID paragraph
      ENUM-ID paragraph }
```

12.5.1 PROGRAM-ID Paragraph

The PROGRAM-ID paragraph indicates that this IDENTIFICATION DIVISION starts a program definition. It also specifies the name that identifies the program and the program attributes COMMON and INITIAL.

The program prototype format is used, in NetCOBOL for .NET, as a means of specifying the interface to unmanaged programs. The system uses a special custom attribute, the DLLIMPORT attribute, to define other information about the unmanaged program such as calling convention, character set, and names of the DLL and entry point to be invoked (defined in the constructor of the custom attribute).

Format 1

```
PROGRAM-ID.  { program-name-1 [AS literal-1]
             { program-name-literal

[ IS { COMMON    } PROGRAM ]
     { INITIAL   }
     { RECURSIVE }

[ CUSTOM-ATTRIBUTE-phrase ] .
```

Format 2

Program prototype definition

```
PROGRAM-ID. program-name-2 [AS literal-1 ]
    IS PROTOTYPE
    CUSTOM-ATTRIBUTE-phrase.
```

Syntax Rules

- Rules for Both Formats

  Literal-1 should be an alphanumeric literal or national literal. However, it should not be a hexadecimal-alphanumeric literal or a hexadecimal-national literal.

- Rules for Format 1

  1. An external name (Literal-1) cannot be specified when the program is contained within another program.
  2. The COMMON clause can only be specified when the program is contained within another program.
  3. The RECURSIVE clause can be written in an external program only.

- Rules for Format 2

  In the CUSTOM-ATTRIBUTE-phrase you must specify the custom-attribute-name used to define the DLLIMPORT attribute. See the topic "CUSTOM-ATTRIBUTE Phrase" for details.

General Rules

- Rules for Format 1

  1. Program-name-1 defines the name of the program declared in this program definition. If literal-1 is specified, program-name-1 is the internal name for the program and literal-1 is the external name. When literal-1 is not specified, program-name-1 is used as both the internal name and the external name.
  2. When the INITIAL phrase is specified, the program is given the initial attribute. When a program with the initial attribute set is called the program is started in its initial state. Programs with the initial attribute set are called initial programs.
  3. Specify the COMMON clause to give a program the common attribute. A program having the common attribute is called a "common program." A common program can be called from programs other than the program containing the common program.
  4. When the RECURSIVE clause is specified, all nested programs inherit the recursive attribute. The program can call itself and it can be called when in the active state. However, nested programs cannot be called directly from other nested programs. Unless the RECURSIVE clause is inherited or specified directly, a program cannot be called again when it is already active.

- Rules for Format 2

  1. Program-name -2 defines a program-prototype-name. The program-prototype-name is a name used to identify the interface of a program. It is used when you want to call unmanaged code (code outside .NET managed code) with the same interface as this definition.
  2. Details of the program to be invoked are defined by using the DllImportAttribute custom attribute. For example, properties of the DllImportAttribute specify the name of the library containing the entry point, the calling convention to be used, and so on. For further details about DllImportAttribute, see the description of this attribute in Microsoft's .NET reference. If the entry point name is not specified in the instantiation of the DllImportAttribute, then program-name-2 is assumed to be the entry point name. See the topic "CUSTOM-ATTRIBUTE Clause" on page for details on specifying custom attributes.
  3. The Data Division of the program prototype definition can only contain a LINKAGE SECTION.
  4. The Procedure Division of the program prototype definition can contain only the PROCEDURE DIVISION header.

12.5.2 CLASS-ID Paragraph

The CLASS-ID paragraph indicates that this IDENTIFICATION DIVISION starts a class definition. It also specifies the name used to identify the class, the inheritance of the class, and any attributes.

Format

CLASS-ID. class-name-1 [AS literal-1]

[IS PARTIAL]

[INHERITS class-name-2]

[CUSTOM-ATTRIBUTE-phrase].

Syntax Rules

See the topic "CUSTOM-ATTRIBUTE Phrase" for details of this phrase.

1. Literal-1 should be an alphanumeric literal or national literal. However, it should not be a hexadecimal-alphanumeric literal or a hexadecimal-national literal.

2. Class-name-2 should be a name of a class specified in the REPOSITORY paragraph of this source element.
3. Class-name-2 should not be the name of the class declared in this class definition.
4. Class-name-2 should not inherit directly or indirectly from class-name-1.
5. Class-name-2 should be a class that is accessible and neither a sealed class nor a private class. Note: Sealed and private classes can be created in other .NET languages, but not in NetCOBOL for .NET.
6. Since class-name-.ASSEMBLY is a compiler reserved word used to define assembly custom attributes, the following rules are also applied if .ASSEMBLY is specified as Literal-1:
    a. Only the class name and CUSTOM-ATTRIBUTE phrase can be specified in the CLASS-ID paragraph.
    b. In the remainder of the class definition, only the CLASS-ID paragraph and CONFIGURATION SECTION of the ENVIRONMENT DIVISION can be specified.
7. Since class-name-.MODULE is a compiler reserved word used to define module custom attributes, the following rules are also applied if .MODULE is specified as a Literal-1:
    a. Only the class name and the CUSTOM-ATTRIBUTE phrase can be specified in the CLASS-ID paragraph.
    b. In the remainder of the class definition, only the CLASS-ID paragraph and the CONFIGURATION SECTION of the ENVIRONMENT DIVISION can be specified.

General Rules

1. Class-name-1 names the class declared in this class definition. When literal-1 is specified, it is the external name of the class, and class-name-1 is the internal name. When literal-1 is not specified, class-name-1 is used as both the internal and external name.
2. The INHERITS clause specifies the name of the class inherited by class-name-1 according to the following rules:
    a. In a class definition, one definition can inherit another class definition.
    b. The inheriting definition gains all the methods from the class that is inherited, including methods that the inherited class may have inherited.
    c. The inheriting class also contains all the data that was defined in the class that is inherited, including data that that class may have inherited. This data will be part of objects created from the inheriting class.
    d. Method names defined in the inherited class become user words in the inheriting class, and can be referred to just as if the method were defined within the inheriting class. The other user-defined words in the inherited definition are not inherited. These user-defined words can be used in the inheriting class as if they were not defined in the inherited class.
3. When the INHERITS phrase is omitted, the System.Object class is implicitly inherited.
4. If class-name-2 is a class that is abstract, all abstract methods and abstract properties contained in class-name-2 must be overridden within this class definition.
5. The PARTIAL clause treats this class as a partial class.
6. The class that specifies .ASSEMBLY or .MODULE for Literal-1 cannot inherit any references because it does not represent a real class.
7. The class that specifies .ASSEMBLY for Literal-1 it is used to specify custom attributes associated with assembly in which it is included.
8. The class that specifies .MODULE for Literal-1 it is used to specify the custom attributes associated with the module in which it is included.

12.5.3 STATIC Paragraph

The STATIC paragraph indicates that this IDENTIFICATION DIVISION starts a static definition.

Format

STATIC.

Syntax Rules

Only one static definition should be specified for a class.

12.5.4 OBJECT Paragraph

The OBJECT paragraph indicates that this IDENTIFICATION DIVISION starts an object definition.

Format

OBJECT. [IMPLEMENTS { interface-name-1} ...] .

Syntax Rules

1. Interface-name-1 should be the name of an interface specified in the REPOSITORY paragraph of the class definition of this source element.
2. Only one object definition should be specified for a class. When there is a static definition, the object definition, if present, must follow the static definition.

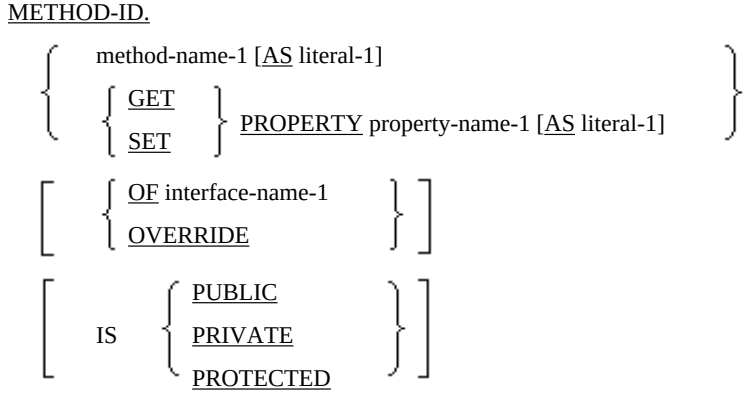
General Rules

The IMPLEMENTS clause specifies the interface(s) supported by the object, indicating that all methods and properties contained in interface- name-1 are implemented explicitly or implicitly in this object definition.

12.5.5 METHOD-ID Paragraph

The METHOD-ID paragraph indicates that this IDENTIFICATION DIVISION starts a method definition. The METHOD-ID paragraph also specifies the method's name and attributes.

Format



Syntax Rules

See the topic "CUSTOM-ATTRIBUTE Phrase" for details of this phrase.

1. Literal-1 should be an alphanumeric literal or national literal. However, it should not be a hexadecimal-alphanumeric literal or a hexadecimal-national literal.
2. OVERRIDE can only be specified for an object method.

3. When OVERRIDE is specified, the METHOD-ID paragraph should define a method with the same method signature as a method that is inherited from the parent class. On method overriding, whether the method signatures match is determined by the following rules:

    a. The external method name must be the same.

    b. The number of formal parameters is the same. (In NetCOBOL for .NET, formal parameters are specified in the USING phrase of the PROCEDURE DIVISION header.)

    c. Corresponding formal parameters receive the arguments in the same way. (In NetCOBOL for .NET, the way of receiving arguments is specified using the BY VALUE or BY REFERENCE phrases of the PROCEDURE DIVISION header.) If BY VALUE is specified, the corresponding parameter also receives the argument by value. If BY REFERENCE is specified (explicitly or implicitly), the corresponding parameter receives the argument by reference.

    d. The data types of corresponding formal parameters match. If a formal parameter is a CLR (Common Language Runtime) type, both types should be the same as each other. If a formal parameter is a COBOL-specific data type, each specification of USAGE/PICTURE (including the picture character-string of edited data items)/ SIGN/BLANK WHEN ZERO/JUSTIFIED should match.

    e. Both returning items have the same data type. (In NetCOBOL for .NET, the returning item is specified in the RETURNING phrase of the PROCEDURE DIVISION header. If there is no RETURNING phrase, the returned item is assumed to be the void type.) Whether returned items are the same is judged as in d above.

4. Only methods that are marked as virtual can be overridden. Note that all instance methods generated from NetCOBOL for .NET are virtual. Methods with System.ParamArrayAttribute cannot be overridden.

5. When OVERRIDE is specified, the accessibility specifiers (i.e. PUBLIC, PRIVATE OR PROTECTED) of the method being defined and the method being overridden should be the same.

6. The OF phrase can be specified only for method definitions contained within object definitions.

7. When the OF phrase is specified, interface-name-1 should contain a method that has the same method signature as the method defined by this METHOD-ID paragraph. The rules to determine that the method signatures match are the same as those defined in syntax rule 3.

8. Property-name-1 should not be a level 01 or 77 data item of the WORKING-STORAGE SECTION in either a static definition or object definition.

9. If the GET phrase is specified, the PROCEDURE DIVISION header should not contain the USING phrase. It should have the RETURNING phrase.

10. If the SET phrase is specified, one USING BY VALUE parameter should be specified in the PROCEDURE DIVISION header. The PROCEDURE DIVISION header should not have the RETURNING phrase.

11. When both a get property method and a set property method are defined in a class for the same property-name-1, whether implicitly from an ancestor class or explicitly, the methods should match in the following:

    - Both accessor methods must be specified as instance methods in the OBJECT section or both must be specified as static methods in the STATIC section

    - Accessibility specifier

    - Type of the returned parameter from the GET method and type of the formal parameter to the SET method using the same rules as defined in syntax rule 3.

12. The compiler reserves the method name "NEW" for the constructor.
    Therefore, when NEW is specified for a method name, the following rules in addition to the above rules are applied:

    a. If the constructor is defined as a static method, it is a static constructor, also known as a class constructor.

    b. If the constructor is defined as an object method, it is defined as an instance constructor.

    c. Only the method name and the CUSTOM-ATTRIBUTE phrase can be specified for the METHOD-ID paragraph of a static constructor.

d. In the instance constructor method name paragraph, the only specifications that are permitted, apart from the method name, are:

   - CUSTOM-ATTRIBUTE clause
   - Accessibility specifiers

13. The following method names are reserved. These names cannot be used as external names in any method definition.

    - .ctor
    - .cctor

    The name .cctor is the name used when defining a static constructor (the NEW method in the static definition).

    You should also avoid using method names, and user-defined words, with the following prefixes as the compiler uses these prefixes for automatically generated methods:

    - get_
    - add_
    - raise_
    - set_
    - remove_
    - op_

14. The accessibility of a method that implements an interface should be PUBLIC, but if the implemented interface-name is specified explicitly in the OF phrase, the accessibility of the method should be PRIVATE.

General Rules

1. If this method definition is included in an object definition, methods or properties declared with this method definition are virtual. If the method definition is included in a static definition, methods and properties declared with this method definition are static.
2. Method-name-1 defines the name of the method declared in this method definition.
   When literal-1 is specified, it is the external name of the method, and method-name-1 is the internal name. When literal-1 is not specified, method-name-1 is used as both the internal and external name.
3. Property-name-1 defines the name of the property declared in this method definition.
   When literal-1 is specified, it is the external name of the property, and property-name-1 is the internal name. When literal-1 is not specified, property-name-1 is used as both the internal and external name.
4. The OVERRIDE clause indicates that this method will replace an inherited method that is virtual. The OVERRIDE clause cannot be specified for a static method because references to static methods are not polymorphic and the class is named explicitly for calls to static methods.
5. The external name of the method can be used with an object identifier to invoke this method.
6. If a property has both an external name and an internal name, the property is accessed using its external name outside of the source file that defines the property, but must be accessed using its internal name within the same source file.
7. If the name of a data item is duplicated in the data divisions of both the method and the static or object definition, then it is necessary to use qualification to access the data item in the static or object definition.
8. If the GET phrase is specified, this method will be a get property method with the name property-name-1 or literal-1.
9. If the SET phrase is specified, this method will be a set property method with the name property-name-1 or literal-1.
10. The OF phrase specifies the interface definition containing the method that you would like to implement.
    When the interface name is omitted, it is assumed that the method implements the interface of the same name as the method, when it is part of a class that implements more than one interface.
11. PUBLIC, PRIVATE, and PROTECTED specify the accessibility of the method. The default is PUBLIC, if the accessibility is not explicitly specified.

12. Multiple methods with the same name can be declared if each method signature is unique. On invoking one of these methods, the most appropriate method to be invoked is selected by the format of the arguments; i.e. the number, order, and data type or kind (by value/by content/by reference) of each argument. This mechanism is called "(method) overloading". The following is an example of method overloading:

```
*> (a) definition of METHOD1 (no parameters)
 METHOD-ID. METHOD1.
 PROCEDURE  DIVISION.
*>     :
 END METHOD METHOD1.

*> (b) definition of another METHOD1 (has one parameter)
 METHOD-ID. METHOD1.                    *>...[1]
 DATA       DIVISION.
 LINKAGE    SECTION.
 01 P1      PIC S9(9) COMP-5.
 PROCEDURE DIVISION USING BY VALUE P1.
*>           :
 END METHOD METHOD1.
*>
 METHOD-ID. MAIN.
 PROCEDURE  DIVISION.
  *> invoke METHOD1 with no parameters
     INVOKE SELF "METHOD1"              *>...[2]
 END METHOD MAIN.
```

[1] METHOD1 (b) has a different signature from METHOD1 (a), therefore it can be defined.

[2] METHOD1 (a) will be invoked, because no argument is specified in this INVOKE statement.

13. COBOL-specific types are regarded as the same in the signature. So, you cannot overload methods that differ from others only in the details of COBOL-specific type parameters.

14. Overloaded methods must differ in the CLR type of their receiving arguments - they may not differ only in the type of their returned value. You cannot overload property methods.

15. 15. You cannot overload static constructors.

12.5.6 DELEGATE-ID Paragraph

The DELEGATE-ID paragraph indicates that this IDENTIFICATION DIVISION starts a delegate definition. It also defines the name of the delegate.

Format

DELEGATE-ID. delegate-name-1 [AS literal-1].

Syntax Rules

Literal-1 should be an alphanumeric literal or national literal. However, it should not be a hexadecimal-alphanumeric literal or a hexadecimal-national literal.

General Rules

1. Delegate-name-1 defines the name of the delegate declared in this delegate definition.
   When literal-1 is specified, it is the external name of the delegate, and delegate-name-1 is the internal name. When literal-1 is not specified, delegate-name-1 is used as both the internal and external name.

2. The Data Division of the delegate definition can include only a LINKAGE SECTION.

3. The PROCEDURE DIVISION of the delegate definition can contain only the PROCEDURE DIVISION header.

12.5.7 INTERFACE-ID Paragraph

The INTERFACE-ID paragraph indicates that this IDENTIFICATION DIVISION starts an interface definition. It also defines the name of the interface.

Format

INTERFACE-ID. interface-name-1 [AS literal-1]

[INHERITS {interface-name-2}...]

[CUSTOM-ATTRIBUTE phrase]

[IS PARTIAL].

Syntax Rules

1. Literal-1 should be an alphanumeric literal or national literal. However, it should not be a hexadecimal-alphanumeric literal or a hexadecimal-national literal.
2. Interface-name-2 is the name of an interface that is specified in the REPOSITORY paragraph of this source element.
3. Interface-name-2, must not inherit interface-name-1 directly or indirectly.

General Rules

1. Interface-name-1 defines the name of the interface declared in this interface definition.
   When literal-1 is specified, it is the external name of the interface.
2. The INHERITS phrase specifies the name of the interface that is inherited by interface-name-1 in accordance with the following rules:
   a. An interface definition can inherit one or more other interface definitions.
   b. The inheriting definition inherits all the method definitions in the interfaces that were inherited, including definitions that those interfaces may have inherited.
3. The PARTIAL phrase specifies the interface as a partial interface.
4. The DATA DIVISION of methods or properties that are contained in an interface definition can include only a LINKAGE SECTION.
5. Only an IDENTIFICATION DIVISION can be included in the PROCEDURE DIVISION of methods or properties contained in an interface definition.

12.5.8 ENUM-ID Paragraph

The ENUM-ID paragraph indicates that this IDENTIFICATION DIVISION starts an enumeration definition. It also defines the name of the enumeration.

Format

ENUM-ID. enum-name-1 [AS literal-1]

[INHERITS class-name-1]

[CUSTOM-ATTRIBUTE phrase].

Syntax Rules

1. Literal-1 should be an alphanumeric literal or national literal. However, it should not be a hexadecimal-alphanumeric literal or a hexadecimal-national literal.

2. Class-name-1 shall be of one of the following classes and be specified in the REPOSITORY paragraph of this source element.
   - System.Byte(byte)
   - System.Int16(short)
   - System.Int32(int)
   - System.Int64(long)

General Rules

1. Enum-name-1 defines the name of the enumeration declared in this enumeration definition.
   When literal-1 is specified, it is the external name of the enumeration.
2. The INHERITS phrase specifies the type of the elements of the ENUM.
3. If the INHERITS phrase is omitted, the class is assumed to be System.Int32.
4. An enumeration definition must not include a PROCEDURE DIVISION.

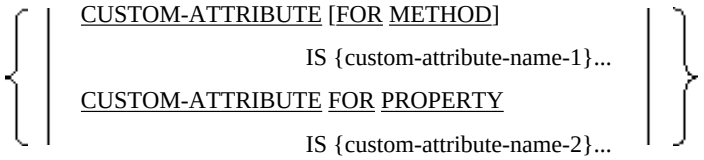
12.5.9 CUSTOM-ATTRIBUTE Phrase

The CUSTOM-ATTRIBUTE phrase provides the means of associating custom attributes with a particular definition or entry.

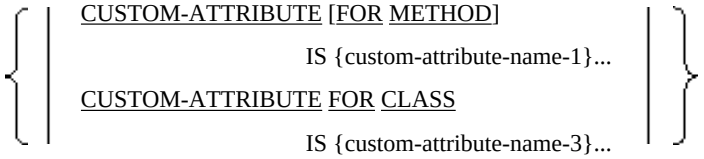
Format 1

CUSTOM-ATTRIBUTE IS {custom-attribute-name-1} ...

Format 2



Format 3



Syntax Rules

1. The CUSTOM-ATTRIBUTE phrase can only be specified in the following places:
   - CLASS-ID paragraph
   - PROGRAM-ID (except internal programs)
   - METHOD-ID
   - Data description entry in the LINKAGE SECTION
   - INTERFACE-ID paragraph.
   - ENUM-ID

- Data items whose type is a CLR type that are declared in the WORKING-STORAGE SECTION of static or object definitions.

2. Custom-attribute-name-1 and custom-attribute-name-2 are the names of custom attributes defined using the CUSTOM-ATTRIBUTE clause of the SPECIAL-NAMES paragraph.
3. When the attribute class that defines custom-attribute-name-1 does not permit multiple specifications for the same definition, you cannot specify more than one custom-attribute-name from that attribute class.
4. Custom-attribute-name-1 should be specified where its attribute is valid.
5. Format 2 can only be used in METHOD-ID paragraphs with GET or SET PROPERTY specifications.
6. For properties that have both GET and SET methods, then both methods should either have the same FOR PROPERTY custom attribute phrase or neither method should have a FOR PROPERTY custom attribute phrase.
7. Format 3 can only be used in the PROGRAM-ID paragraph which does not include the PROTOTYPE clause.

General Rules

- Rules for Format 2

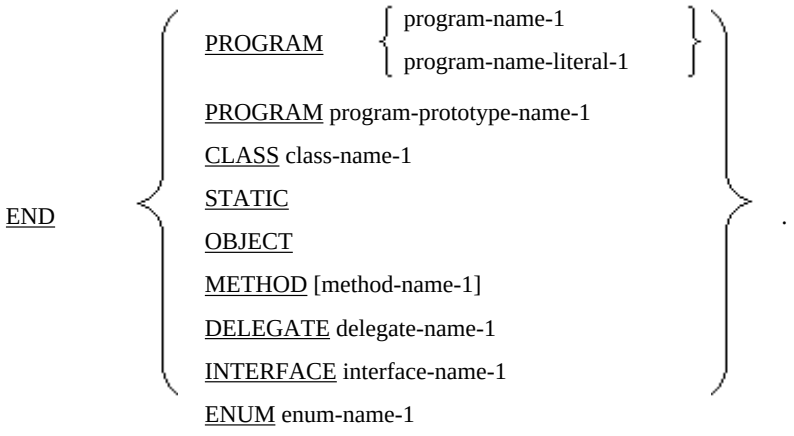
1. The CUSTOM-ATTRIBUTE phrase without a FOR phrase or the CUSTOM-ATTRIBUTE phrase with a FOR METHOD phrase is used to specify custom attributes for the method.
2. The CUSTOM-ATTRIBUTE phrase with a FOR PROPERTY phrase is used to specify custom attributes for the property.

- Rules for Format 3

1. CUSTOM-ATTRIBUTE phrases that do not specify a FOR or FOR METHOD are used to specify the custom attribute given to the method.
2. CUSTOM-ATTRIBUTE phrases that specify FOR CLASS are used to specify the custom attribute given to the class.

12.5.10 END Markers

Format



Syntax Rules

1. An END marker should be written at the end of each source unit. It indicates that containing units continue from this point.
2. Program-name-1 must match the program-name specified in the PROGRAM-ID paragraph.
3. Program-prototype-name-1 must match the program-prototype-name specified in the PROGRAM-ID paragraph.
4. Class-name-1 must match the class-name specified in the CLASS-ID paragraph.

5. Method-name-1 must match the method name specified in the METHOD-ID paragraph. When a PROPERTY specification is written in the METHOD-ID paragraph, you should omit method-name-1.
6. Delegate-name-1 must match the delegate name specified in the DELEGATE-ID paragraph.
7. Interface-name-1 must match the interface name specified in the INTERFACE-ID paragraph.
8. Enum-name-1 must match the enumeration name specified in the ENUM-ID paragraph.

General Rules

1. The END markers indicate the end of each source unit.
2. An END marker of a source unit contained within another source unit should either be followed by another END marker, if the containing source unit is also terminated, or by the first statement of a following source unit.
3. The first statement of the next compilation unit, if any, should only follow an END marker of a source unit that is not contained within another source unit.

12.6 Environment Division

The ENVIRONMENT DIVISION consists of two sections: the CONFIGURATION SECTION and the INPUT-OUTPUT SECTION. This section describes the specific support in these sections for NetCOBOL for .NET.

12.6.1 CONFIGURATION SECTION

Format

CONFIGURATION SECTION.

[Source-computer paragraph]

[Object-computer paragraph]

[Special-names paragraph]

[Repository paragraph]

Syntax Rules

1. The CONFIGURATION SECTION can be specified in program, program prototype, class, delegate, interface and enumeration definitions. It cannot be specified in static, object and method definitions.
2. The SOURCE-COMPUTER and OBJECT-COMPUTER paragraphs cannot be specified in class, delegate, interface or enumeration definitions.

12.6.1.1 SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph defines a number of program-specific behaviors and names. For NetCOBOL for .NET the definition of the custom-attribute-name has been added.

Format

SPECIAL-NAMES.

[{ function-name-1 clause }...]

[{ function-name-2 clause } ...]

[{ function-name-3 clause } ...]

[{ ALPHABET clause } ...]

[{ CLASS clause } ...]

[CURRENCY SIGN clause]

[{ CUSTOM-ATTRIBUTE clause } ...]

[DECIMAL-POINT IS COMMA clause]

[POSITIONING UNIT clause]

[PRINTING MODE clause]

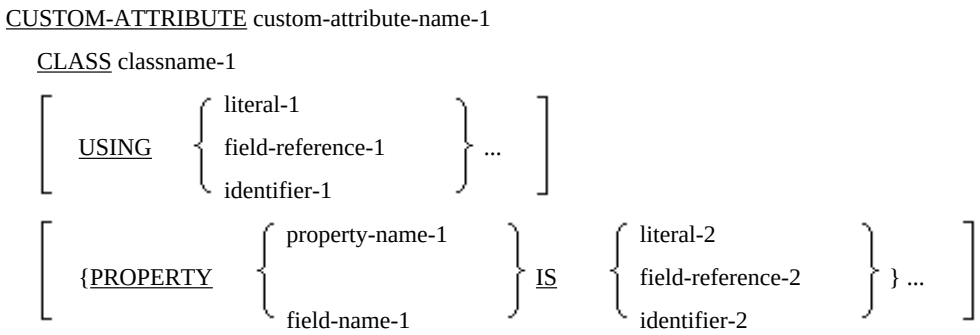
[{ SYMBOLIC CHARACTERS clause } ...]

[{ SYMBOLIC CONSTANT clause } ...]

12.6.1.2 CUSTOM-ATTRIBUTE Clause

The CUSTOM-ATTRIBUTE clause defines a custom attribute that is referred to in the source unit by the name custom-attribute-name.

Format



Syntax Rules

1. Identifer-1 and identifier-2 should be a field reference, function-identifier or TYPE OF operator.
2. Class-name-1 should be a class name of an attribute class (a class that inherits from the System.Attribute class). However, it should not be the name of the class "System.ParamArrayAttribute".
3. Class-name-1 should be specified in the REPOSITORY paragraph of this source unit.
4. Property-name-1 should be a property name of the class specified by class-name-1.
5. Field-name-1 should be a field name of the class specified by class-name-1.
6. Field-reference-1 and field-reference-2 must reference fields that contain literals (defined at compile time - as the custom attribute information populates metadata that is written at compile time).
7. Function-identifier-1 and function-identifier-2 should be one of the following functions:
   - FUNCTION ENUM-OR
   - FUNCTION ENUM-AND
   - FUNCTION ENUM-NOT
8. Literal-1, field-reference-1, and function-identifier-1 should match the parameter of the constructor of the class specified in class-name-1.
9. Literal-2, field-reference-2, and function-identifier-2 should match the type of the corresponding field-name-1 and property name-1.

General Rules

1. Literal-1, field-reference-1, and function-identifier-1 are restricted by the types of the constructor parameters. The parameters of the constructor of class-name-1 should be one of the following managed types:

System.Byte (uint8)	System.Char (char)

System.Int16 (int16)	System.Boolean (Boolean)
System.Int32 (int32)	System.Object (object)
System.Int64 (int64)	System.String (string)
System.Single (float32)	enumeration type
System.Double (float64)	System.Type

2. Field-name-1 and property-name-1 should be one of the following types:

System.Byte (uint8)	System.Char (char)
System.Int16 (int16)	System.Boolean (Boolean)
System.Int32 (int32)	System.Object (object)
System.Int64 (int64)	System.String (string)
System.Single (float32)	enumeration type
System.Double (float64)	System.Type

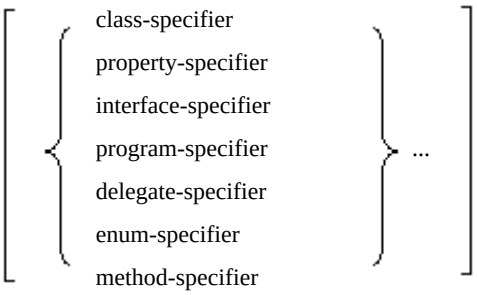
12.6.1.3 REPOSITORY Paragraph

The REPOSITORY paragraph contains information about names that are defined in separately compiled programs. This includes class names, property names, field names, interface names, program prototype names, delegate names, method identifiers, and enumeration names.

The REPOSITORY paragraph can be included in class definitions, program definitions, program prototype definitions, and delegate definitions.

Format

REPOSITORY.

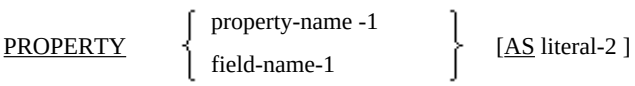


Where the specifiers have the following formats:

class-specifier

CLASS class-name-1 [AS literal-1]
[EXPANDS class-name-2
USING specification]

property-specifier



interface-specifier

INTERFACE interface-name-1 [AS literal-3]
[EXPANDS interface-name-2

USING specification]

program-specifier

PROGRAM program-prototype-name-1 [AS literal-4]

delegate-specifier

DELEGATE delegate-name-1 [AS literal-5]

[EXPANDS delegate-name-2

USING specification]

enum-specifier

ENUM enum-name-1 [AS literal-6]

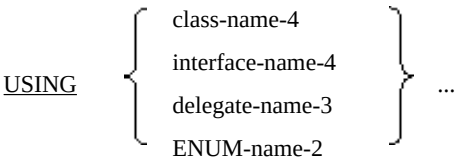
method-specifier

METHOD method-identifier-1 [AS literal-7]

[EXPANDS method-identifier-2

USING specification]

USING specification



Syntax Rules

1. No class-name-1, property-name-1, interface-name-1, program-prototype-name-1, delegate-name-1, enum-name-1, or method-name-1 should be specified more than once in the REPOSITORY paragraph.
2. Literal-1, literal-2, literal-3, literal-4, literal-5, literal-6, and literal-7 should be either alphanumeric literals or national literals. However, they should not be hexadecimal-alphanumeric literals or hexadecimal-national literals.
3. Literal-1, literal-2, literal-3, literal-4, literal-5, and literal-6 should not be character strings that start with an asterisk (*).
4. When the REPOSITORY paragraph is described in a class definition, it should not contain the class-name of the class that is being defined.
5. Literal-1 in the class-specifier can indicate an array type by including brackets ([]) after the class name string in the literal. For more detail, refer to "External Names" on page 688.
6. If you specify a generic class, a generic interface, a generic delegate, or a generic method for a class-specifier, an interface-specifier, a delegate-specifier, or a method-specifier respectively, literal-1 can indicate a generic type by including angular brackets (<>) after the name of the type or method. For more detail, refer to "External Names" on page 688.

Rules specific to class-specifier

1. Class-name-2, class-name-4, interface-name-4, delegate-name-3, and ENUM-name-2 must be names declared in the same REPOSITORY paragraph.
2. Class-name-2 must be the name specified by the class with the type parameter.
3. The number of names in the USING specification must match the number of type parameters of the class specified by class-name-2 in the EXPANDS specification.

Rules specific to property-specifier

Property-specifier specifies the alias name for referencing object properties or fields that may be used throughout the scope of the containing environment division. If the PROPERTY phrase is specified:

a. If you define properties or fields within the scope of the environment division, the combination of their defined names and referred names should be unique by conforming to that of property-name-1/field-name-1 and literal-2.

Example

```
REPOSITORY.
    PROPERTY  P1
    PROPERTY  P2  AS  "P".                          ...[1]
       :
    01 P1 PIC N(1) PROPERTY.             *> OK
    01 P2 PIC N(1) PROPERTY.             *> ERROR   ...[2]
    01 P  PIC N(1) PROPERTY.             *> OK
       :
    METHOD-ID. GET PROPERTY P1.          *> OK
       :
    METHOD-ID. GET PROPERTY P2.          *> ERROR   ...[3]
       :
    METHOD-ID. GET PROPERTY P2 AS "P".   *> OK      ...[4]
```

[1] The property-specifier specifies that property or field referred to as "P2" is defined using the name "P".

[2] and [3] Here a property referred to as "P2" is defined using the name "P2" and so these statements cause compilation errors.

[4] Here the property referred to as "P2" is defined using the name "P" so does not cause a conflict and no compiler error will be produced.

b. If literal-2 is not specified, there must be information defined explicitly or implicitly for the property, property-name-1, or field, field-name-1, in the source unit containing this CONFIGURATION SECTION or in the metadata that is part of one of the classes or interfaces that are declared in this REPOSITORY paragraph.

c. If literal-2 is specified, there must be information defined explicitly or implicitly for the property/field, literal-2, in the source unit containing this CONFIGURATION SECTION or in the metadata that is part of one of the classes or interfaces that are declared in this REPOSITORY paragraph.

Rules specific to interface-specifier

1. Interface-name-2, class-name-4, interface-name-4, delegate-name-3, and ENUM-name-2 must be names declared in the same REPOSITORY paragraph.
2. Interface-name-2 must be the name specified by the interface with the type parameter.
3. The number of names in the USING specification must match the number of type parameters with the interface specified by interface-name-2 in the EXPANDS specification.

Rules specific to delegate-specifier

1. Delegate-name-2, class-name-4, interface-name-4, delegate-name-3, and ENUM-name-2 must be names declared in the same REPOSITORY paragraph.
2. The name specified for delegate-name-2 must be the same as that specified by the delegate with the type parameter.
3. The number of names in the USING specification must match the number of type parameters in the delegate specified by delegate-name-2 in the EXPANDS specification.

Rules specific to method-specifier

1. Method-identifier-2, class-name-4, interface-name-4, delegate-name-3, and ENUM-name-2 must be names declared in the same REPOSITORY paragraph.

2. Method-identifier-2 must be the name specified by the method for the type parameter.
3. The number of names in the USING specification must match the number of type parameters in the method specified by method-identifier-2 in the EXPANDS specification.

General Rules

1. If AS is specified, the word before AS is used as the internal name, and the literal specified after AS is used as the external name.
2. The literals of the AS specification, except in the case of property-specifier, are fully qualified external names which combine an item name, namespace modifier, and enclosing type modifier .

Rules specific to class-specifier

1. A class-specifier declares class-name-1 as a class used within the source unit containing the ENVIRONMENT DIVISION.
2. If class-name-1 specifies a generic class, it can only be used in the EXPANDS specification of another class-specifier.
3. The type specified for the USING specification is used as type arguments to the generic class specified by class-name-2. The types must satisfy any constraints on class-name-2 for the corresponding type parameters.
4. For class-specifiers that use EXPANDS, class-name-1 refers to the instantiation of the generic class specified by class-name-2 with the type arguments in the USING specification.

Rules specific to property-specifier

A property-specifier declares property-name-1 or field-name-1 to be the name of the property or field used within the source unit containing the environment division.

Rules specific to interface-specifier

1. An interface-specifier declares interface-name-1 to be the name of an interface used within the source unit containing the environment division.
2. If interface-name-1 specifies a generic interface, it can only be used in the EXPANDS specification of another interface-specifier.
3. The types specified for the USING specification are used as type arguments to the generic interface specified by interface-name-2. The types must satisfy any constraints on interface-name-2 for the corresponding type parameters.
4. For interface-specifiers that use EXPANDS, interface-name-1 refers to the instantiation of the generic interface specified by interface-name-2 with the type arguments in the USING specification.

Rules specific to program-specifier

1. A program-specifier declares program-prototype-name-1 to be the name of the program prototype used within the source unit containing the environment division.
2. The program-specifier and the program prototype definition referenced by program-prototype-name must be defined in the same compilation group.

Rules specific to delegate-specifier

1. The delegate-specifier declares delegate-name-1 to be the name of a delegate used within the source unit containing the environment division.
2. If delegate-name-1 specifies a generic delegate, it can only be used in the EXPANDS specification of another delegate-specifier.
3. The types specified for the USING specification are used as type arguments to the generic delegate specified by delegate-name-2. The types must satisfy any constraints on delegate-name-2 for the corresponding type parameters.

4. For delegate-specifiers that use EXPANDS, delegate-name-1 refers to the instantiation of the generic delegate specified by delegate-name-2 with the type arguments in the USING specification.

Rules specific to ENUM-specifier

The ENUM-specifier declares ENUM-name-1to be the name of the ENUM used within the source unit containing the environment division.

Rules specific to method-specifier

1. A method-specifier declares method-identifier-1 to be the name of a method used within the source unit containing the environment division .
2. If method-identifier-1 specifies a generic method, it can only be used in the EXPANDS specification of another method-specifier.
3. The types specified for the USING specification are used as type arguments to the generic method specified by method-identifier-2. The types must satisfy any constraints on method-identifier-2 for the corresponding type parameters.
4. For method-specifiers that use EXPANDS, method-identifier-1 refers to the instantiation of the generic method specified by method-identifier-2 with the type arguments in the USING specification.

12.6.2 INPUT-OUTPUT Section

Format

INPUT-OUTPUT SECTION.

[File-control paragraph]

[I/O control paragraph]

Syntax Rules

The INPUT-OUTPUT SECTION can be specified in program definitions, static definitions, object definitions, and method definitions except method definitions in interface definitions.

12.6.3 I-O-CONTROL Paragraph

There are no NetCOBOL-for-.NET-specific formats, though there are some restrictions as specified below.

Syntax Rules

1. Do not specify the:
   - APPLY MULTICONVERSATION-MODE clause
   - APPLY SAVED-AREA clause
   - MULTIPLE FILE TAPE clause, or
   - RERUN clause,

   in the I-O-CONTROL paragraph.
2. Do not specify the USAGE OBJECT REFERENCE phrase for data items that are subordinate to data-name-1 of the APPLY SAVED-AREA clause.

12.7 Data Division

12.7.1 FILE SECTION

1. The FILE SECTION can only be specified in a program definition, static definition, object definition or method definition except in method definitions in interface definitions.
2. In the FILE SECTION of a static definition, object definition, or method definition, the following clauses cannot be used:
   - LABEL RECORDS
   - VALUE OF
   - DATA RECORDS.
3. The REPORT IS phrase is not supported.

12.7.2 SCREEN SECTION

The SCREEN SECTION is not permitted under .NET COBOL.

12.7.3 REPORT SECTION

The REPORT SECTION is not permitted under .NET COBOL.

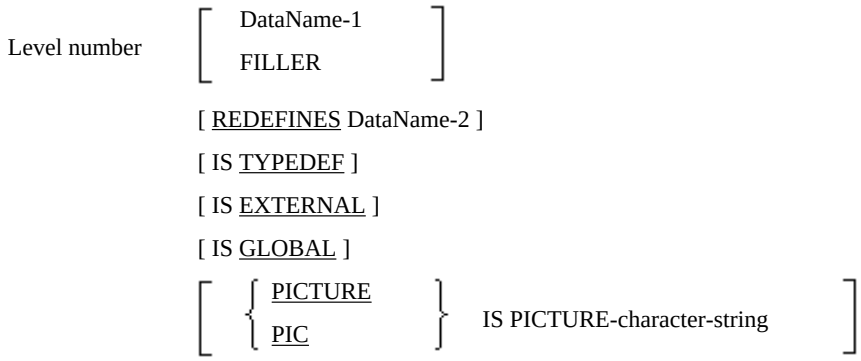
12.7.4 LINKAGE SECTION

Syntax Rules

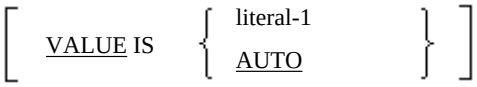
1. A data item defined in the LINKAGE SECTION can be referenced from the PROCEDURE DIVISION only when the following criteria are met:
   a. It is an operand of the USING or RETURNING phrase of the PROCEDURE DIVISION header.
   b. It is subordinate to an operand of the USING or RETURNING phrase of the PROCEDURE DIVISION header.
   c. It REDEFINES or RENAMES an item that meets the above criteria.
   d. It is subordinate to an item defined by c).
   e. It is a condition-name or index-name associated with a data item that meets one of the above criteria.
2. If a data item defined in the LINKAGE SECTION is referenced, its data-name should not duplicate other data-names defined in linkage sections within the COBOL source unit.

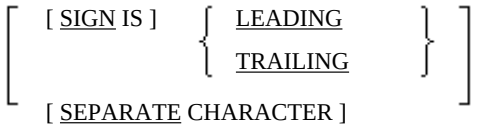
12.7.5 Data Description Entry

Format

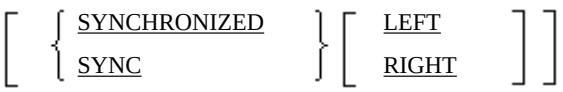


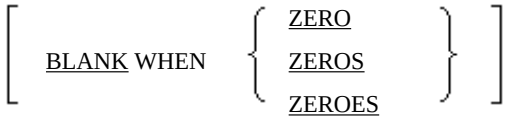
[USAGE phrase]

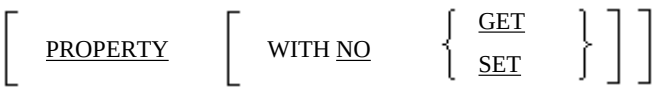




[OCCURS phrase]





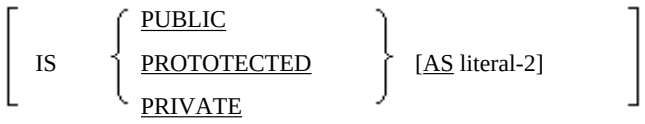


[TYPE type-name-1]

[CHARACTER TYPE clause]

[PRINTING POSITION clause]

[CUSTOM-ATTRIBUTE phrase]



.

Syntax Rules

1. Level 01 or 77 data items that are declared in the WORKING-STORAGE SECTION of a static or object definition cannot be unnamed or FILLER and their names cannot be the same as the names of other 01 or 77 level items in the static or object definitions.
2. The elements of an enum definition are made up of level 01 elementary items declared in the WORKING-STORAGE SECTION. The items must have a VALUE clause and no other clause.
3. For an enum definition, the VALUE clause contains either a constant or the word AUTO. The constant must be a number that is in the valid range of values for the type specified by the INHERITS clause of the ENUM-ID paragraph.

12.7.5.1 CUSTOM-ATTRIBUTE Phrase

For the syntax of this phrase see Format 1 of the "CUSTOM-ATTRIBUTE Phrase" in the IDENTIFICATION DIVISION section.

Syntax Rules

1. The CUSTOM-ATTRIBUTE phrase may only be specified in the following places:
   - Data description entries in the LINKAGE SECTION
   - CLR types that are declared in the WORKING-STORAGE SECTION of static or object definitions.

2. Custom-attribute-name-1 must map to an attribute class that specifies an attribute that applies to the item definition that includes the CUSTOM-ATTRIBUTE phrase.
   When the attribute-class is "System.ParamArrayAttribute" class, the data item containing the CUSTOM-ATTRIBUTE phrase must also be:
   - a. In the LINKAGE SECTION of a method (other than a GET or SET PROPERTY method).
   - b. Specified in the last USING phrase in the PROCEDURE DIVISION header.
   - c. A one-dimensional array.
3. When the attribute class that defines custom-attribute-name-1 does not permit multiple specifications for the same definition, you cannot specify more than one custom-attribute-name from that attribute class.

12.7.5.2 CHARACTER TYPE Clause

For the definition of the CHARACTER TYPE clause see "CHARACTER TYPE Clause" under "Data Description Entry" in "Chapter 5. Data Division".

Syntax Rules

1. The CHARACTER TYPE clause can only be specified for items in the DATA DIVISION of a program or class definition.
2. A CHARACTER TYPE clause with a DEPENDING ON phrase must not be used in the DATA DIVISION of a static or object definition.

12.7.5.3 EXTERNAL Clause

The EXTERNAL clause is as defined in " EXTERNAL Clause" under "Data Description Entry" in "Chapter 5. Data Division" with the addition of the following rule.

Format

IS EXTERNAL

Syntax Rules

1. The EXTERNAL clause should not be used in the Data Division of either a static or object definition.
2. Do not specify the EXTERNAL clause for a data item with USAGE OBJECT REFERENCE.
3. Do not specify the EXTERNAL clause for a BINARY-CHAR/SHORT/LONG/DOUBLE data item.

12.7.5.4 GLOBAL Clause

The GLOBAL clause is as defined in "GLOBAL Clause" under "Data Description Entry" in "Chapter 5. Data Division" with the addition of the following rule.

Format

IS GLOBAL

Syntax Rules

1. The GLOBAL clause should not be used in the DATA DIVISION of either a static or object definition.
2. Do not specify the GLOBAL clause for a data item with USAGE OBJECT REFERENCE in the LINKAGE SECTION.

12.7.5.5 OCCURS Clause

The OCCURS clause is as defined in "OCCURS Clause" under "Data Description Entry" in "Chapter 5. Data Division" with the addition of the following rule.

Syntax Rules

Do not use a Boolean item or an object reference in the KEY IS clause.

12.7.5.6 PRINTING POSITION Clause

The PRINTING POSITION clause is as defined in "PRINTING POSITION Clause" under "Data Description Entry" in "Chapter 5. Data Division" with the addition of the following rule.

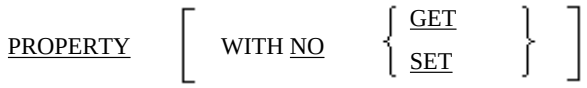
Syntax Rules

The PRINTING POSITION clause can only be used in the DATA DIVISION of a program or class definition.

12.7.5.7 PROPERTY Clause

The PROPERTY clause indicates that this data item is considered to be a property of this object for which get property/set property methods will be automatically generated unless otherwise specified.

Format



Syntax Rules

1. The PROPERTY clause can only be specified in the WORKING-STORAGE SECTION of a static definition or object definition.
2. The PROPERTY clause cannot be specified for data items that are subordinate to the OCCURS clause.
3. The PROPERTY clause cannot be specified for data items that are subordinate to the TYPEDEF clause.
4. The PROPERTY clause can only be used for 01 or 77 level, elementary items.
5. When you want to use the PROPERTY clause and the TYPE clause at the same time, the type to which the TYPE clause refers must be defined as an elementary item that does not infringe on syntax rule 4.
6. When the NO SET phrase is omitted, the data item should be one of:
   - national item
   - numeric item,
   - object reference item

General Rules

1. If the GET phrase is not specified, the PROPERTY clause will generate a method defined for the static or object that includes it.
   When the usage of the data item is an object reference or an index, this method's implicit definition is as follows:
   (The "data-name" and "data-description" are replaced by the name and description of the item carrying the PROPERTY clause.)

```
METHOD-ID. GET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01 LS-data-name  data-description.
PROCEDURE DIVISION RETURNING LS-data-name.
paragraph-name.
    SET LS-data-name TO data-name.
    EXIT METHOD.
END METHOD.
```

2. When the class of the data item is alphanumeric or national, this method's implicit definition is:

```
METHOD-ID. GET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01 LS-data-name  data-description.
PROCEDURE DIVISION RETURNING LS-data-name.
paragraph-name.
    MOVE data-name TO LS-data-name(1:)
    EXIT METHOD.
END METHOD.
```

3. Otherwise the implicit definition of the method is:

```
METHOD-ID. GET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01 LS-data-name  data-description.
PROCEDURE DIVISION RETURNING LS-data-name.
paragraph-name.
    MOVE data-name TO LS-data-name
    EXIT METHOD.
END METHOD.
```

   Here, LS-data-name has the same data description entries as this data item, including subordinate clauses, except for the following clauses:

   - INDEXED BY clause
   - PROPERTY clause
   - VALUE clause
   - REDEFINES clause
   - CUSTOM-ATTRIBUTE clause
   - Accessibility specifier clause

4. If the SET phrase is not specified, the PROPERTY clause generates a method for the static or object that includes it. When the usage of the data item is an object reference, the implicit definition of the method is as follows: ("data-name" and "data-description" are replaced by the name and description of the item carrying the PROPERTY clause)

```
METHOD-ID. SET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01 LS-data-name  data-description.
PROCEDURE DIVISION USING BY VALUE LS-data-name.
paragraph-name.
    SET data-name TO LS-data-name
    EXIT METHOD.
END METHOD.
```

5. When the class of the data-item is alphanumeric or national, the implicit definition of the method is:

```
METHOD-ID. SET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01 LS-data-name  data-description.
PROCEDURE DIVISION USING BY VALUE LS-data-name.
paragraph-name.
    MOVE LS-data-name TO data-name(1:)
    EXIT METHOD.
END METHOD.
```

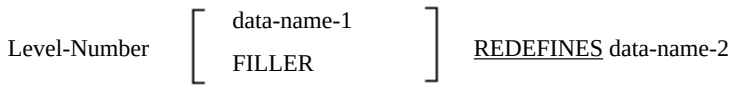
Otherwise the implicit definition of the method is:

```
METHOD-ID. SET PROPERTY data-name.
DATA DIVISION.
LINKAGE SECTION.
01 LS-data-name  data-description.
PROCEDURE DIVISION USING BY VALUE LS-data-name.
paragraph-name.
    MOVE LS-data-name TO data-name
    EXIT METHOD.
END METHOD.
```

12.7.5.8 REDEFINES Clause

The REDEFINES clause is as defined in "REDEFINES Clause" under "Data Description Entry" in "Chapter 5. Data Division" with the addition of the following rule.

Format



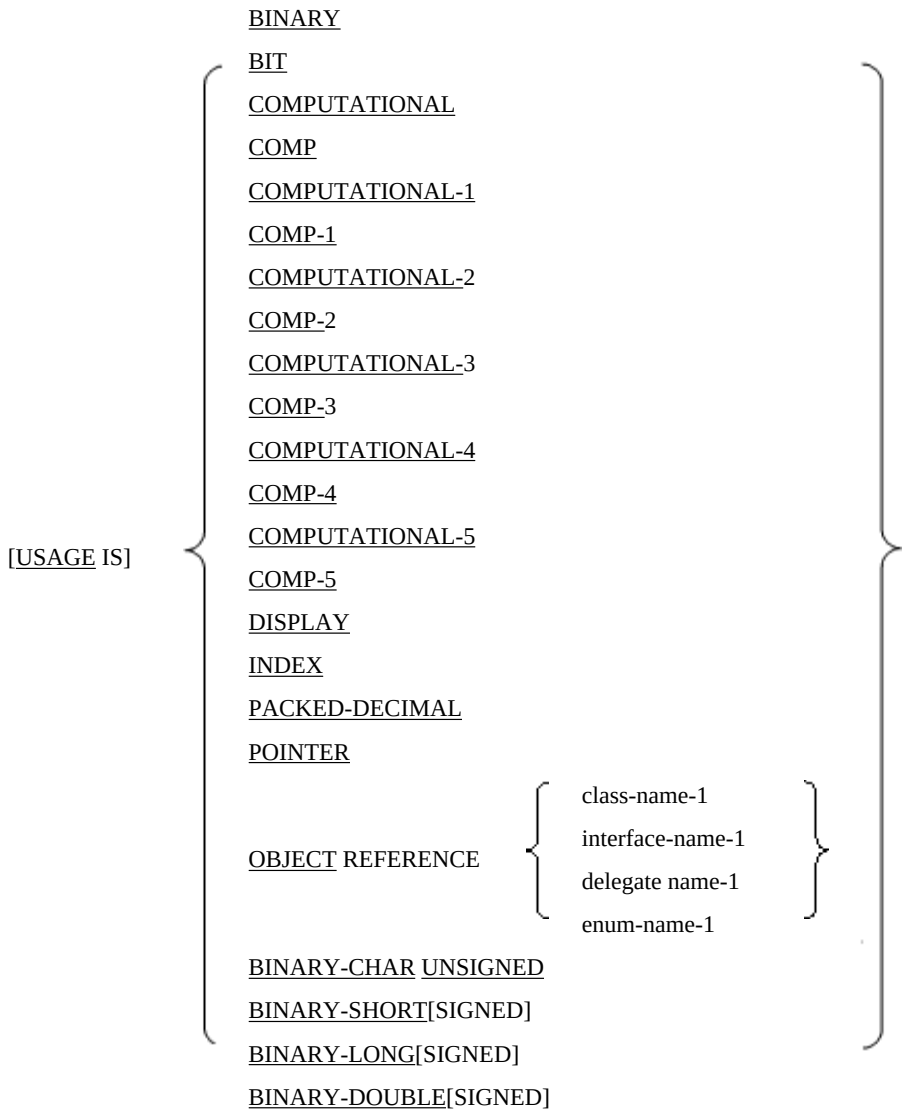
Syntax Rules

1. The REDEFINES clause cannot be used for a data item declared as an object reference.
2. The REDEFINES clause cannot be used with a BINARY-CHAR/SHORT/LONG/DOUBLE data item.
3. The data description entry for data-name-2 should not include a data item to be used as an object reference.
4. The data description entry for data-name-2 should not include a BINARY-CHAR/SHORT/LONG/DOUBLE data item.
5. When data-name-2 is a CLR-type data item, the data type of data-name-1 must be the corresponding COBOL-specific type. See Table 12.3 for mapping of CLR data types to COBOL data types.

12.7.5.9 USAGE Clause

The USAGE clause is as defined in "USAGE Clause" under "Data Description Entry" in "Chapter 5. Data Division" with the addition of the following rule.

Format



Syntax Rules

1. BLANK WHEN ZERO, JUSTIFIED, PICTURE, and SYNCHRONIZED should not be specified on data items whose usage is INDEX or OBJECT REFERENCE.
2. USAGE OBJECT REFERENCE can only be specified for elementary items with a level number of 01 or 77.
3. USAGE OBJECT REFERENCE can only be used in the WORKING-STORAGE, LINKAGE and LOCAL-STORAGE sections.
4. Class-name-1 should be declared in a class specifier in the source unit that contains the data description, or be the name of the class that contains the data description.
5. When class-name-1 is one of the following classes, the data description can only be in:
   - The WORKING-STORAGE SECTION or LOCAL-STORAGE SECTION of a program definition
   - The WORKING-STORAGE SECTION of a method definition.

   System.Byte System.Int16
   System.Int32 System.Int64
   System.Single System.Double
   System.Char

6. You cannot specify an OBJECT REFERENCE for class-name-1 when class-name-1 is System.Void.
7. Interface-name-1 should be declared in an interface specifier in the source unit that contains the data description.
8. Delegate-name-1 should be declared in a delegate specifier in the source unit that contains the data description.
9. Enum-name-1 should be declared in an enum specifier in the source unit that contains the data description.

General Rules

1. A data description that includes the USAGE OBJECT REFERENCE clause allocates sufficient storage to maintain the object reference.
2. A data item with USAGE OBJECT REFERENCE is used to refer to objects dynamically.
3. The following table shows the correspondence of the basic CLR data types to the COBOL data types.

Table 12.1 CLR Data Type Mapping to COBOL Data Types

CLR Type	COBOL Data Type	PICTURE
int8	OBJECT REFERENCE "System.SByte"	(N/A)
unsigned int8	BINARY-CHAR UNSIGNED	(N/A)
int16	COMP-5	S9(4)
	BINARY-SHORT SIGNED	(N/A)
unsigned int16	OBJECT REFERENCE "System.UInt16"	(N/A)
int32	COMP-5	S9(9)
	BINARY-LONG SIGNED	(N/A)
unsigned int32	OBJECT REFERENCE "System.UInt32"	(N/A)
int64	COMP-5	S9(18)
	BINARY-DOUBLE SIGNED	(N/A)
unsigned int64	OBJECT REFERENCE "System.UInt64"	(N/A)
float32	COMP-1	(N/A)
float64	COMP-2	(N/A)
char	DISPLAY	N
decimal	OBJECT REFERENCE "System.Decimal"	(N/A)
boolean	OBJECT REFERENCE "System.Boolean"	(N/A)
object	OBJECT REFERENCE "System.Object"	(N/A)
string	OBJECT REFERENCE "System.String"	(N/A)

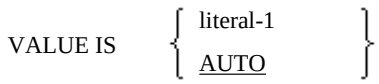
The "System.xxxx " entries in the above table are actually the external class names of class names defined in class specifiers in the REPOSITORY paragraph.

4. For an item to be treated as CLR type data, it should meet all the following requirements. If it misses on any one of the requirements it is considered to be a COBOL-specific type.
    a. Be a data item defined in either the WORKING-STORAGE, LINKAGE or LOCAL-STORAGE sections. (But in the case of a program definition, data items defined in the LINKAGE SECTION are considered to be of COBOL-specific type, except for data items declared with USAGE OBJECT REFERENCE.)
    b. Be an elementary item of either 01 or 77 level.
    c. Does not redefine other items, and is not redefined by other items.
    d. Must not be renamed.
    e. Data description must not contain the EXTERNAL clause.
    f. Must not be a SQL host variable.
    g. Must have one of the following data descriptions:
        - OBJECT REFERENCE,
        - BINARY-CHAR UNSIGNED,
        - BINARY-SHORT SIGNED,
        - BINARY-LONG SIGNED,
        - BINARY-DOUBLE SIGNED,
        - S9(4) COMP-5,
        - S9(9) COMP-5,
        - S9(18) COMP-5,
        - COMP-1,
        - COMP-2,
        - PIC N
    h. Must not have either the CHARACTER TYPE or PRINTING POSITION clause.
5. The support for USAGE POINTER items is tailored for the .NET environment. See the online NetCOBOL for .NET User's Guide for a description of using pointers with the .NET product.

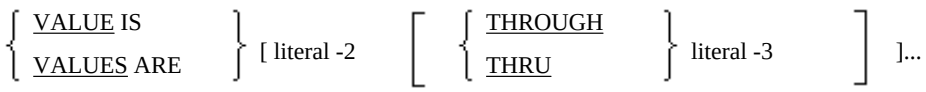
12.7.5.10 VALUE Clause

The VALUE clause is as defined in "VALUE Clause" under "Data Description Entry" in "Chapter 5. Data Division" with the addition of the following rule.

Format 1



Format 2



Syntax Rules

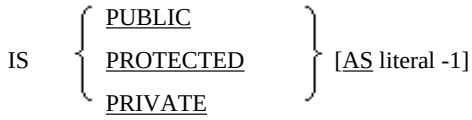
1. The VALUE clause cannot be specified for a data item whose usage is OBJECT.
2. AUTO can only be specified on elements of an enum definition.

General Rules

1. A data item, which is neither redefined nor included in any redefinition and for which the USAGE OBJECT REFERENCE clause is specified, is initialized to reference the NULL object if its data is reference type data. When the data is value type data, the initial value is undefined.
2. When AUTO is specified, the value of the element is allotted automatically.
3. When AUTO is specified, if the VALUE phrase is on the first element defined in the enumeration, the initial value of the element is 0. Otherwise the value of the item with AUTO specified is the value of the previous item plus 1.

12.7.5.11 Accessibility Specifier Phrase (PUBLIC/PRIVATE/PROTECTED)

Format



Syntax Rules

1. The accessibility specifier phrase can only be specified for CLR type data items declared in the WORKING-STORAGE SECTION of object or static definitions or in the LINKAGE SECTION of a method definition.
2. PRIVATE is the only accessibility that can be specified for LINKAGE SECTION items.
3. literal -1 is a nonnumeric literal or national language non-numeric literal. However, it should not be a hexadecimal nonnumeric literal or national hexadecimal nonnumeric literal.
4. When literal-1 is specified for a WORKING-STORAGE SECTION item, literal-1 must not be the same as any external names of fields or properties declared in this class.
5. When literal-1 is specified for a LINKAGE SECTION item, literal-1 must not be the same as any external names of data items defined in that LINKAGE SECTION.
6. If the access-modifier phrase is omitted, PRIVATE is assumed, unless there is a PROPERTY clause, in which case the access modifier is assumed to be PUBLIC.

General Rules

1. When the accessibility specifier is specified on an item with a PROPERTY phrase, the accessibility specifier applies to the property.
2. When the AS phrase is specified, literal-1, becomes the external name of this field, property or LINKAGE SECTION data.
3. For PUBLIC fields and properties access is not restricted.
4. PROTECTED fields and properties can be accessed from within the class in which it is defined or classes that inherit from that class.
5. PRIVATE fields and properties (specified explicitly or implicitly) can only be accessed by the class in which they are defined. LINKAGE SECTION data always has PRIVATE accessibility and can only be accessed from the method definition in which it is included.

12.8 Procedure Division

The PROCEDURE DIVISION of a method or program contains the procedure code to be executed.

The PROCEDURE DIVISION in a static or object definition contains the methods to be invoked.

12.8.1 PROCEDURE DIVISION Header

Format 1

PROCEDURE DIVISION with DECLARATIVES

```
PROCEDURE DIVISION
        [using-phrase]
        [RETURNING data-name-2 ]
        [RAISING {class-name-1} ...].
[DECLARATIVES.
        {section-name SECTION.
use-statement.
        [paragraph-name. [sentence] ...] ...}
END DECLARATIVES. ]
{section-name SECTION.
        [paragraph-name. [sentence] ...] ...}...
```

Format 2

PROCEDURE DIVISION without DECLARATIVES

```
PROCEDURE DIVISION
        [using-phrase]
        [RETURNING data-name-2 ]
        [RAISING { class-name-1}...].
          [paragraph-name. [sentence] ...] ...}...
```

Format 3

PROCEDURE DIVISION with no procedure code

```
PROCEDURE DIVISION
      [using-phrase]
      [RETURNING data-name-2 ]
      [RAISING { class-name-1} ...].
```

Format 4

PROCEDURE DIVISION for static, object and interface definitions

```
PROCEDURE DIVISION.
      [{ method definition} ...]
```

The using-phrase

```
USING  { [ BY REFERENCE ] { data-name-1} ... }
       { BY VALUE { data-name-1} ...         }  ...
```

Syntax Rules

- Rules for Formats 1, 2, and 3

  1. Data-name-1 must be defined as a level-number 01 or 77 item in the LINKAGE SECTION. A user-defined word must not appear more than once as data-name-1. The data description of data-name-1 must not include the REDEFINES clause.
     Data-name-1 may be redefined by another data item in the LINKAGE SECTION.
  2. If BY VALUE is used with data-name-1 then data-name-1 must be specified either as a CLR type data item or a numeric item.
  3. Data-name-2 must be defined as a level-number 01 or 77 item in the LINKAGE SECTION. A user-defined word must not appear more than once as data-name-2. The data description of data-name-2 must not include the REDEFINES clause.
     Data-name-2 may be redefined by another data item in the LINKAGE SECTION.
  4. Data-name-2 should not be the same as data-name-1.
  5. Class-name-1 should be the name of a class specified in the REPOSITORY paragraph.
  6. The BY VALUE phrase cannot be used in the PROCEDURE DIVISION header of a program definition.
  7. USING and RETURNING cannot be specified to find the procedure division of static constructors.
  8. RETURNING cannot be specified to find the procedure division of instance constructors.

- Rules for Format 3

  Format 3 can be used only in a program prototype definition or delegate definition.

- Rules for Format 4

  Format 4 can be used only in a static definition, object definition or interface definition.

General Rules

- Rules for Formats 1 and 2

  Execution starts at the first statement in the PROCEDURE DIVISION, excluding the declaratives. The statements are executed in the order that they appear during compilation except when the rules of execution specify another order.

- Rules for Formats 1, 2, and 3

  1. The using-phrases identify the names of the formal parameters used by the method or program. The arguments to be passed to the parameters are identified by one of the following invoking source elements:
     - using-phrase of CALL statement
     - using-phrase of INVOKE statement
     - Operand(s) specified for in-line invocation of a method
  2. Correspondence of arguments and parameters is based on their positions in their respective lists.
  3. Both the BY REFERENCE and the BY VALUE specifications are effective until another BY REFERENCE or BY VALUE specification is defined for a subsequent parameter. When neither BY REFERENCE nor BY VALUE is specified for the first parameter, BY REFERENCE is assumed.
  4. Data-name-1 is a formal parameter of a method or program.
  5. Data-name-2 is a name used in the method or program to return a result to the invoking code.
  6. The initial value of data-name-2 is undefined.
  7. When the value of an argument is passed BY REFERENCE the invoked code executes as if the data item in the LINKAGE SECTION occupies the same area as the argument passed by the invoking code.

8. When the value of an argument is passed BY CONTENT the invoked code executes as if the invoking code allocated a memory area for the LINKAGE SECTION during invocation-time initialization. This memory does not occupy the same address as the argument passed by the invoking code.

   If a program (no program-specifier) is activated, the allocated record has the same length as those of the argument. The argument is transcribed to the record without being converted. The invoked side handles the record as if it were an argument passed BY REFERENCE.

   If a method or program (specified by program-specifier) is activated, the allocated record has the same length as that of the record in the LINKAGE SECTION (formal parameter). The argument is used as a sending operand, and the LINKAGE SECTION record as a receiving operand, in one of the following statements:

   - COMPUTE statement when the argument is numeric.
   - SET statement when the argument is an index or object reference
   - Otherwise a MOVE statement

   The invoked code handles the record as if it were an argument passed BY REFERENCE.

9. When the argument is passed BY VALUE, the invoking code executes processing as if a record in the LINKAGE SECTION had been allocated during invocation-time initialization.

   - The allocated record has the same description as that of the LINKAGE SECTION record (formal parameter). The argument is used as a sending operand in a COMPUTE statement without the ROUNDED phrase, and the LINKAGE SECTION record is the receiving operand.

10. References to data-name-1 and data-name-2, in the invoked code, are always resolved using the descriptions of these items in the LINKAGE SECTION.

11. Class-name-1 is a class that inherits from the class of the exception object. An exception may be generated at the EXIT statement of the invoked code.

12.8.2 Common Statement Rules

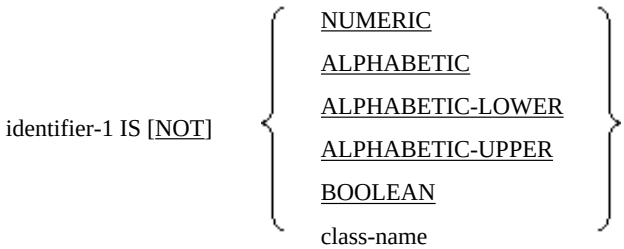
This section describes the NetCOBOL-for-.NET-specific rules that are common to all statements.

12.8.2.1 Conditional Expressions

12.8.2.1.1 Class Condition

The class condition is used to examine the class of a data item.

Format



General Rules

1. A class condition inspects the contents of identifier-1 and returns a truth value according to the rules of class inspection. The rules for returning TRUE are shown in the table below.

Table 12.2 Rules for Class Conditions

Class	Rules for Class Condition to be TRUE
NUMERIC	Identifier-1 contains only the digits 0 to 9, or the digits 0to 9 with operational signs. (*1)
ALPHABETIC	Identifer-1 contains uppercase letters A to Z and spaces, or lowercase letters a to z and spaces, or uppercase and lowercase letters and spaces.
ALPHABETIC-LOWER	Identifier-1 contains lowercase letters a to z and spaces.
ALPHABETIC-UPPER	Identifier-1 contains uppercase letters A to Z and spaces.
BOOLEAN	Identifier-1 contains only the Boolean characters 0 and 1.
Character class name	Identifier-1 contains characters specified in the CLASS clause of the SPECIAL-NAMES paragraph.

*1: For details see rule 6.

2. Identifier-1 must be one of the following:
   - A data item of usage display
   - An alphanumeric or national language function-identifier
   - A packed-decimal data item.
3. For the NUMERIC test identifer-1 cannot be any of the following:
   - An alphabetic data item
   - A group item containing signed numeric data items
   - A national data item
   - A national edited data item
   - A Boolean data item
4. For the ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, and the class-name tests, identifier-1 cannot be any of the following:
   - A numeric data item
   - A national data item
   - A national edited data item
   - A Boolean data item
5. For the BOOLEAN test, identifier-1 cannot be any of the following:
   - A numeric data item
   - An alphabetic data item
   - A national data item
   - A national edited data item
6. The NUMERIC test returns TRUE under any of the following conditions:
   a. When the data description of identifier-1 does not specify an operational sign and there is no operational sign in the contents of identifier-1 or, if identifier-1 is defined as a packed-decimal item, identifier-1 contains hexadecimal F in the sign position.

b. When the data description of identifier-1 does specify an operational sign, identifier-1 contains only numeric digits and:

- If the SIGN clause includes the SEPARATE CHARACTER phrase, the sign part is "+" or "-" expressed in the standard data format, or,
- If the SIGN clause does not include the SEPARATE CHARACTER phrase for a display numeric, the sign part of the sign-carrying digit is hexadecimal 4, 5, or 3, or,
- If identifier-1 is a packed-decimal item the sign position contains hexadecimal C, D or F.

7. When NOT and a class condition key word are combined to form a class condition, the expression with NOT has a truth value that is opposite to that of the expression without NOT.

12.8.2.1.2 Type Condition

Type conditions are used to inspect class inheritance and interface implementation.

Format

identifier-1 IS [NOT] { class-name-1 | interface-name-1 | delegate-name-1 | enum-name-1 }

General Rules

1. Type conditions inspect the contents of identifier-1 and return a truth value according to the rules of type inspection. The rules for returning TRUE are shown in the table below.

Table 12.3 Rules for Type Conditions

Type	Rules for Type Condition to be TRUE
class-name-1	The object reference stored in identifier-1 inherits from class-name-1. or The type of the object reference stored in identifier-1 is class-name-1
interface-name-1	The object reference stored in identifier-1 implements the interface interface-name-1.
delegate-name-1	The object reference stored in identifier-1 is the delegate delegate-name-1.
enum-name-1	The object held stored in identifier-1 is the enumeration enum-name-1.

2. Identifier-1 must be an object reference.
3. When NOT and a type condition are combined to form a class condition, the expression with NOT has a truth value that is opposite to that of the expression without NOT.

12.8.2.2 Comparisons

The tables on the following pages show the combinations of data items that can be compared.

Table 12.4 Part (i). Comparison of Data (No entry => the comparison cannot be made)

Subject of condition Object of condition	Alphabetic, Alphanumeric, Alphanumeric edited, Numeric edited	Alphanumeric function	Display numeric	Packed decimal	Numeric function, Integer function	Floating-point	National, National function, National edited
Alphabetic Alphanumeric Alphanumeric literal Hexadecimal-alphanumeric literal Alphanumeric edited Numeric edited	[1]	[1]	[1]	-	-	-	-
Alphanumeric function	[1]	[1]	[1]	-	-	-	-
Display numeric	[1]	[1]	[2]	[2]	[2]	[2]	-
Packed decimal	-	-	[2]	[2]	[2]	[2]	-
Numeric literal Hexadecimal literal	[1]	-	[2]	[2]	[2]	[2]	-
Arithmetic function Integer function	-	-	[2]	[2]	[2]	[2]	-
Floating-point Floating-point literal	-	-	[2]	[2]	[2]	[2]	-
National National function National edited ALL national literal National hexadecimal literal	-	-	-	-	-	-	[3]
Boolean ALL Boolean literal	-	-	-	-	-	-	-
Group	[1]	[1]	[1]	[1]	-	[1]	[1]
Index	-	-	-	-	-	-	-
Object reference	-	-	-	-	-	-	-
ZERO	[1]	[1]	[2]	[2]	[2]	[2]	-
SPACE	[1]	[1]	[1]	-	-	-	[3]
HIGH-VALUE LOW-VALUE	[1]	[1]	[1]	-	-	-	[3]
QUOTE symbolic-character	[1]	[1]	[1]	-	-	-	-
EXCEPTION-OBJECT	-	-	-	-	-	-	-
NULL	-	-	-	-	-	-	-
SELF	-	-	-	-	-	-	-
Enum value	-	-	-	-	-	-	-

Subject of condition Object of condition	Alphabetic, Alphanumeric, Alphanumeric edited, Numeric edited	Alphanumeric function	Display numeric	Packed decimal	Numeric function, Integer function	Floating-point	National, National function, National edited
Enum function							

[1]: Nonnumeric comparison

[2]: Numeric comparison

[3]: National nonnumeric comparison

Table 12.5 Part (ii). Comparison of Data (No entry => the comparison cannot be made)

Subject of condition Object of condition	Boolean	Group (no type/ weak typing)	Index	Object reference	EXCEPTION-OBJECT	NULL	SELF	Enum value Enum function
Alphabetic Alphanumeric Alphanumeric literal Hexadecimal-alphanumeric litera Alphanumeric edited Numeric edited	-	[1]	-	-	-	-	-	-
Alphanumeric function	-	[1]	-	-	-	-	-	-
Display numeric	-	[1]	-	-	-	-	-	-
Packed decimal	-	[1]	-	-	-	-	-	-
Numeric literal Hexadecimal literal	-	[1]	-	-	-	-	-	-
Arithmetic function Integer function	-	[1]	-	-	-	-	-	-
Floating-point Floating-point literal	-	[1]	-	-	-	-	-	-
National National function National edited ALL national literal National hexadecimal literal	-	[1]	-	-	-	-	-	-
Boolean ALL Boolean literal	[4]	-	-	-	-	-	-	-
Group	-	[1]	-	-	-	-	-	-

Subject of condition Object of condition	Boolean	Group (no type/ weak typing)	Index	Object reference	EXCEPTION-OBJECT	NULL	SELF	Enum value Enum function
Index	-	-	[5]					
Object reference	[6] *3	-	-	[6] *1	[6] *1	[6] *1	[6] *1	[6] *2
ZERO	[4]	[1]	-	-	-	-	-	-
SPACE	-	[1]	-	-	-	-	-	-
HIGH-VALUE LOW-VALUE	-	[1]	-	-	-	-	-	-
QUOTE symbolic-character	-	[1]	-	-	-	-	-	-
EXCEPTION-OBJECT	-	-	-	[6] *1	-	[6]	[6]	-
NULL	-	-	-	[6] *1	[6]	-	[6]	-
SELF	-	-	-	[6] *1	[6]	[6]	-	-
Enum value Enum function	-	-	-	-	-	-	-	[6]

[1]: Nonnumeric comparison

[4]: Boolean comparison

[5]: Index comparison

[6]: Obj. ref. comparison

*1 : Object references to value types can only be compared in the following combinations:

Comparisons of the same ENUM type

Comparisons of the System.Boolean type

This is because object references to value types are not like object references to reference types where the compiler can just compare the contents of the two references; when you use an object reference to a value type (as opposed to using an intrinsic COBOL data type) the compiler does not have sufficient information to obtain a value for comparison - apart from the cases listed above.

*2 : Must be a combination of the same enumeration types.

*3 : Must be a combination of a boolean item or boolean literal for column 1 and System.Boolean type object reference.

12.8.2.3 Rules for Moving (Transcribing) Data

Moving BINARY-CHAR/SHORT/LONG/DOUBLE Data Items

When the sending data is a BINARY-CHAR/SHORT/LONG/DOUBLE data item, the receiving data must be a numeric or numeric edited data item.
For other rules, see the topic titled "Rules for Moving (Transcribing) Data" in the "Common Statement Rules" of "Chapter 6. Procedure Division".

12.8.2.3.1 Parameter Conformance

The following tables show which receiving parameters are considered to conform to which sending parameters, i.e. that it is permissible to define one type as a sending parameter and receive it as another type.

The "passing BY CONTENT" tables also apply to passing BY VALUE.

When conversion from UCS-2 to ACP is indicated in the following tables and notes, it is important to note that:

* If a UCS-2 character value cannot be represented in ACP, an execution time exception will occur.

Table 12.6 Part (i). Parameter Conformance - passing BY REFERENCE.

Formal parameter / Actual argument		Int8	UInt8	Int16	UInt16	Int32	Uint32
CLR Type	BINARY-CHAR UNSIGNED	-	Y	-	-	-	-
	BINARY-SHORT SIGNED	-	-	Y	-	-	-
	BINARY-LONG SIGNED	-	-	-	-	Y	-
	BINARY-DOUBLE SIGNED	-	-	-	-	-	-
	S9(4) COMP-5	-	-	Y	-	-	-
	S9(9) COMP-5	-	-	-	-	Y	-
	S9(18) COMP-5	-	-	-	-	-	-
	COMP-1	-	-	-	-	-	-
	COMP-2	-	-	-	-	-	-
	N(1) [DISPLAY]	-	-	-	-	-	-
	OBJECT REFERENCE	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)

Table 12.7 Part (ii). Parameter Conformance - passing BY REFERENCE.

Formal parameter / Actual argument		Int64	UInt64	Float32	Float64	Char	Boolean	Decimal
CLR type	BINARY-CHAR UNSIGNED	-	-	-	-	-	-	-
	BINARY-SHORT SIGNED	-	-	-	-	-	-	-
	BINARY-LONG SIGNED	-	-	-	-	-	-	-
	BINARY-DOUBLE SIGNED	Y	-	-	-	-	-	-
	S9(4) COMP-5	-	-	-	-	-	-	-
	S9(9) COMP-5	-	-	-	-	-	-	-
	S9(18) COMP-5	Y	-	-	-	-	-	-
	COMP-1	-	-	Y	-	-	-	-
	COMP-2	-	-	-	Y	-	-	-
	N(1) [DISPLAY]	-	-	-	-	Y	-	-
	OBJECT REFERENCE	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)

Table 12.8 Part (iii). Parameter Conformance - passing BY REFERENCE.

Formal parameter / Actual argument		Class	Value (includes ENUM type)	Object	String	Array type	COBOL type
CLR type	BINARY-CHAR UNSIGNED	-	-	-	-	-	-
	BINARY-SHORT SIGNED	-	-	-	-	-	-
	BINARY-LONG SIGNED	-	-	-	-	-	-
	BINARY-DOUBLE SIGNED	-	-	-	-	-	-
	S9(4) COMP-5	-	-	-	-	-	-
	S9(9) COMP-5	-	-	-	-	-	-
	S9(18) COMP-5	-	-	-	-	-	-
	COMP-1	-	-	-	-	-	-
	COMP-2	-	-	-	-	-	-
	N(1) [DISPLAY]	-	-	-	Y(*6)	-	-
	OBJECT REFERENCE	Y(*2)	Y(*2)	Y(*1)	Y(*1)	Y(*2)	

Table 12.9 Part (iv). Parameter Conformance - passing BY REFERENCE.

Formal parameter / Actual argument					Int8	UInt8	Int16	UInt16	Int32	Uint32
COBOL type *8	Embedded CLR type	BINARY-CHAR UNSIGNED			-	Y(*3)	-	-	-	-
		BINARY-SHORT SIGNED			-	-	Y(*3)	-	-	-
		BINARY-LONG SIGNED			-	-	-	-	Y(*3)	-
		BINARY-DOUBLE SIGNED			-	-	-	-	-	-
		S9(4) COMP-5			-	-	Y(*3)	-	-	-
		S9(9) COMP-5			-	-	-	-	Y(*3)	-
		S9(18) COMP-5			-	-	-	-	-	-
		COMP-1			-	-	-	-	-	-
		COMP-2			-	-	-	-	-	-
		N(1) [DISPLAY]			-	-	-	-	-	-
	Numeric	Binary	-	Unsigned	-	-	-	-	-	-
				Signed	-	-	-	-	-	-
			-	Unsigned	-	-	-	-	-	-
				Signed	-	-	-	-	-	-
		Packed decimal	Integer	Unsigned	-	-	-	-	-	-
				Signed	-	-	-	-	-	-
			Non Integer	Unsigned	-	-	-	-	-	-
				Signed	-	-	-	-	-	-
		Display	Integer	Unsigned	-	-	-	-	-	-
				Signed	-	-	-	-	-	-

Formal parameter / Actual argument					Int8	UInt8	Int16	UInt16	Int32	Uint32
			Non integer	Unsigned	-	-	-	-	-	-
				Signed	-	-	-	-	-	-
	Alphabetic data							-	-	-
	Alphanumeric character							-	-	-
	External floating-point							-	-	-
	National				-	-	-	-	-	-
	Boolean				-	-	-	-	-	-
	Numeric edited				-	-	-	-	-	-
	Alphanumeric edited				-	-	-	-	-	-
	National edited				-	-	-	-	-	-
	Group (no type/weakly typed)				-	-	-	-	-	-
	Index				-	-	-	-	-	-
	Pointer data item				-	-	-	-	-	-

Table 12.10 Part (v). Parameter Conformance - passing BY REFERENCE.

Formal parameter / Actual argument					Int64	UInt64	Float32	Float64	Char	Boolean	Decimal
COBOL type *8	Embedded CLR type	BINARY-CHAR UNSIGNED			-	-	-	-	-	-	-
		BINARY-SHORT SIGNED			-	-	-	-	-	-	-
		BINARY-LONG SIGNED			-	-	-	-	-	-	-
		BINARY-DOUBLE SIGNED			Y(*3)	-	-	-	-	-	-
		S9(4) COMP-5			-	-	-	-	-	-	-
		S9(9) COMP-5			-	-	-	-	-	-	-
		S9(18) COMP-5			Y(*3)	-	-	-	-	-	-
		COMP-1			-	-	Y(*3)	-	-	-	-
		COMP-2			-	-	-	Y(*3)	-	-	-
		N(1) [DISPLAY]			-	-	-	-	Y(*3)	-	-
	Numeric	Binary	Integer	Unsigned	-	-	-	-	-	-	-
				Signed	-	-	-	-	-	-	-
			Non integer	Unsigned	-	-	-	-	-	-	-
				Signed	-	-	-	-	-	-	-
		Packed decimal	Integer	Unsigned	-	-	-	-	-	-	-
				Signed	-	-	-	-	-	-	-
			Non Integer	Unsigned	-	-	-	-	-	-	-
				Signed	-	-	-	-	-	-	-
		Display	Integer	Unsigned	-	-	-	-	-	-	-
				Signed	-	-	-	-	-	-	-

Formal parameter / Actual argument					Int64	UInt64	Float32	Float64	Char	Boolean	Decimal
			Non integer	Unsigned	-	-	-	-	-	-	-
				Signed	-	-	-	-	-	-	-
	Alphabetic data				-	-	-	-	-	-	-
	Alphanumeric character				-	-	-	-	-	-	-
	External floating-point				-	-	-	-	-	-	-
	National				-	-	-	-	-	-	-
	Boolean				-	-	-	-	-	-	-
	Numeric edited				-	-	-	-	-	-	-
	Alphanumeric edited				-	-	-	-	-	-	-
	National edited				-	-	-	-	-	-	-
	Group (no type/weakly typed)				-	-	-	-	-	-	-
	Index				-	-	-	-	-	-	-
	Pointer data item				-	-	-	-	-	-	-

Table 12.11 Part (vi). Parameter Conformance - passing BY REFERENCE.

Formal parameter / Actual argument					Class	Value (includes ENUM type)	Object	String	Array type	COBOL type
COBOL type *8	Embedded CLR type	BINARY-CHAR UNSIGNED			-	-	-	-	-	-
		BINARY-SHORT SIGNED			-	-	-	-	-	-
		BINARY-LONG SIGNED			-	-	-	-	-	-
		BINARY-DOUBLE SIGNED			-	-	-	-	-	-
		S9(4) COMP-5			-	-	-	-	-	-
		S9(9) COMP-5			-	-	-	-	-	-
		S9(18) COMP-5			-	-	-	-	-	-
		COMP-1			-	-	-	-	-	-
		COMP-2			-	-	-	-	-	-
		N(1) [DISPLAY]			-	-	-	-	-	-
	Numeric	Binary	Integer	Unsigned	-	-	-	-	-	Y(*4)
				Signed	-	-	-	-	-	Y(*4)
			Non integer	Unsigned	-	-	-	-	-	Y(*4)
				Signed	-	-	-	-	-	Y(*4)
		Packed decimal	Integer	Unsigned	-	-	-	-	-	Y(*4)
				Signed	-	-	-	-	-	Y(*4)
			Non	Unsigned	-	-	-	-	-	Y(*4)

Formal parameter / Actual argument					Class	Value (includes ENUM type)	Object	String	Array type	COBOL type
			Integer	Signed	-	-	-	-	-	Y(*4)
		Display	Integer	Unsigned	-	-	-	-	-	Y(*4)
				Signed	-	-	-	-	-	Y(*4)
			Non integer	Unsigned	-	-	-	-	-	Y(*4)
				Signed	-	-	-	-	-	Y(*4)
	Alphabetic data				-	-	-	Y(*6)	-	Y(*4)
	Alphanumeric character				-	-	-	Y(*6)	-	Y(*4)
	External floating-point				-	-	-	-	-	Y(*4)
	National				-	-	-	Y(*6)	-	Y(*4)
	Boolean				-	-	-	-	-	Y(*4)
	Numeric edited				-	-	-	-	-	Y(*4)
	Alphanumeric edited				-	-	-	-	-	Y(*4)
	National edited				-	-	-	-	-	Y(*4)
	Group (no type/weakly typed)				-	-	-	-	-	Y(*5)
	Index				-	-	-	-	-	Y(*7)
	Pointer data item				-	-	-	-	-	Y(*9)

Notes on the Parameter Conformance BY REFERENCE tables

Y in a cell - types can be combined

Blank cell - types cannot be combined

"Embedded CLR type" means a CLR data type that is defined within a group item - it is not an 01 or 77 level elementary item.

*1 Must be an instance of the appropriate aliased type (System.xxxx).

*2 Must be an instance of the dummy argument's type (delegate or enumeration).

*3 A compiler warning error will be generated when embedded CLR data type items are used. The compiler implicitly converts embedded CLR data types (which are stored as byte arrays) to temporary data items of the appropriate type, and passes these data items to the program or method being invoked.

*4 Information on the custom attribute must match completely with the attribute of the argument.

*5 The length of the data descriptions must be the same.

*6 Static data and object data will generate a W compilation error. For other data, reference conversion is applied as an implicit type conversion. The internal processing is as follows:

When the program is compiled with RCS(UTF8-UCS2)

- Alphanumeric parameters:
  Prior to making the method call the system removes trailing spaces and converts to UCS-2. On return from the call the parameter is converted to UTF-8 and padded with spaces.
- National parameters:
  Trailing spaces are removed before making the call and the returned data is padded with spaces after the call.

When the program is compiled with RCS(ACP-UCS2)

- Alphanumeric parameters:
  Prior to making the method call the system removes trailing spaces and converts to UCS-2. On return from the call the parameter is converted to ACP and padded with spaces.
- National parameters:
  Trailing spaces are removed before making the call and the returned data is padded with spaces after the call.

When the program is compiled with RCS(ACP)

- Alphanumeric parameters:
  Prior to making the method call the system removes trailing spaces and converts to UCS-2. On return from the call the parameter is converted to ACP and padded with spaces.
- National parameters:
  Trailing spaces are removed before making the call and the data converted to UCS-2. The returned data is converted to ACP and padded with spaces after the call.

*7 When the formal parameter is an index data item (USAGE INDEX)

*8 An actual parameter is an alphabetic item, an alphanumeric item or national item, and, when the corresponding formal parameter there are a COBOL type and String type, the COBOL type conforms than the String type.

*9 When the formal parameter is a pointer item (USAGE POINTER)

Priority Order for Matching

An exact type match will take precedence over matches that require the compiler to do some conversions.

Table 12.12 Part (i). Parameter Conformance - passing BY CONTENT.

Formal parameter / Actual argument		Int8	UInt8	Int16	UInt16	Int32	Uint32
CLR type	BINARY-CHAR UNSIGNED	-	Y	Y	Y	Y	Y
	BINARY-SHORT SIGNED	-	-	Y	-	Y	-
	BINARY-LONG SIGNED	-	-	-	-	Y	-
	BINARY-DOUBLE SIGNED	-	-	-	-	-	-
	S9(4) COMP-5	-	-	Y	-	Y	-
	S9(9) COMP-5	-	-	-	-	Y	-
	S9(18) COMP-5	-	-	-	-	-	-
	COMP-1	-	-	-	-	-	-
	COMP-2	-	-	-	-	-	-
	N(1) [DISPLAY]	-	-	-	-	-	-
	OBJECT REFERENCE	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)

Table 12.13 Part (ii). Parameter Conformance - passing BY CONTENT.

Formal parameter / Actual argument		Int64	UInt64	Float32	Float64	Char	Boolean	Decimal
CLR type	BINARY-CHAR UNSIGNED	Y	Y	Y	Y	-	-	-
	BINARY-SHORT SIGNED	Y	-	Y	Y	-	-	-
	BINARY-LONG SIGNED	Y	-	Y	Y	-	-	-
	BINARY-DOUBLE SIGNED	Y	-	Y	Y	-	-	-
	S9(4) COMP-5	Y	-	Y	Y	-	-	-

Formal parameter / Actual argument		Int64	UInt64	Float32	Float64	Char	Boolean	Decimal
	S9(9) COMP-5	Y	-	Y	Y	-	-	-
	S9(18) COMP-5	Y	-	Y	Y	-	-	-
	COMP-1	-	-	Y	Y	-	-	-
	COMP-2	-	-	-	Y	-	-	-
	N(1) [DISPLAY]	-	-	-	-	Y	-	-
	OBJECT REFERENCE	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)

Table 12.14 Part (iii). Parameter Conformance - passing BY CONTENT.

Formal parameter / Actual argument		Class	Value (includes Enum type)	Object	String	Array type	COBOL type
CLR type	BINARY-CHAR UNSIGNED	-	-	Y	-	Y(*4)	-
	BINARY-SHORT SIGNED	-	-	Y	-	Y(*4)	-
	BINARY-LONG SIGNED	-	-	Y	-	Y(*4)	-
	BINARY-DOUBLE SIGNED	-	-	Y	-	Y(*4)	-
	S9(4) COMP-5	-	-	Y	-	Y(*4)	-
	S9(9) COMP-5	-	-	Y	-	Y(*4)	-
	S9(18) COMP-5	-	-	Y	-	Y(*4)	-
	COMP-1	-	-	Y	-	Y(*4)	-
	COMP-2	-	-	Y	-	Y(*4)	-
	N(1) [DISPLAY]	-	-	Y	Y(*8)	Y(*4) (*22)	-
	OBJECT REFERENCE	Y(*2)	Y(*3)	Y	Y(*1)	Y(*3) (*4)	Y(*5)

Table 12.15 Part (iv). Parameter Conformance - passing BY CONTENT.

Formal parameter / Actual argument			Int8	UInt8	Int16	UInt16	Int32	Uint32
COBOL type (*21)	Built-in CLR type	BINARY-CHAR UNSIGNED	-	Y	Y	Y	Y	Y
		BINARY-SHORT SIGNED	-	-	Y	-	Y	-
		BINARY-LONG SIGNED	-	-	-	-	Y	-
		BINARY-DOUBLE SIGNED	-	-	-	-	-	-
		S9(4) COMP-5	-	-	Y	-	Y	-
		S9(9) COMP-5	-	-	-	-	Y	-
		S9(18) COMP-5	-	-	-	-	-	-
		COMP-1	-	-	-	-	-	-
		COMP-2	-	-	-	-	-	-
		N(1) [DISPLAY]	-	-	-	-	-	-

Formal parameter / Actual argument					Int8	UInt8	Int16	UInt16	Int32	Uint32
	Numeric	Binary	Integer	Unsigned	Y(*6)	Y(*6)	Y(*6)	Y(*6)	Y(*6)	Y(*6)
				Signed	Y(*6)	-	Y(*6)	-	Y(*6)	-
			Non Integer	Unsigned	-	-	-	-	-	-
				Signed	-	-	-	-	-	-
		Packed decimal	Integer	Unsigned	Y(*6)	Y(*6)	Y(*6)	Y(*6)	Y(*6)	Y(*6)
				Signed	Y(*6)	-	Y(*6)	-	Y(*6)	-
			Non Integer	Unsigned	-	-	-	-	-	-
				Signed	-	-	-	-	-	-
		Display	Integer	Unsigned	Y(*6)	Y(*6)	Y(*6)	Y(*6)	Y(*6)	Y(*6)
				Signed	Y(*6)	-	Y(*6)	-	Y(*6)	-
			Non Integer	Unsigned	-	-	-	-	-	-
				Signed	-	-	-	-	-	-
	Alphabetic data				-	-	-	-	-	-
	Alphanumeric character				-	-	-	-	-	-
	External floating-point				-	-	-	-	-	-
	National				-	-	-	-	-	-
	Boolean				-	-	-	-	-	-
	Numeric edited				-	-	-	-	-	-
	Alphanumeric edited				-	-	-	-	-	-
	National edited				-	-	-	-	-	-
	Group (no type/weakly typed)				-	-	-	-	-	-
	Index				-	-	-	-	-	-
	Pointer data item				-	-	-	-	-	-

Table 12.16 Part (v). Parameter Conformance - passing BY CONTENT.

Formal parameter / Actual argument			Int64	UInt64	Float32	Float64	Char	Boolean	Decimal
COBOL type (*21)	Built-in CLR type	BINARY-CHAR UNSIGNED	Y	Y	Y	Y	-	-	-
		BINARY-SHORT SIGNED	Y	-	Y	Y	-	-	-
		BINARY-LONG SIGNED	Y	-	Y	Y	-	-	-
		BINARY-DOUBLE SIGNED	Y	-	Y	Y	-	-	-
		S9(4) COMP-5	Y	-	Y	Y	-	-	-
		S9(9) COMP-5	Y	-	Y	Y	-	-	-
		S9(18) COMP-5	Y	-	Y	Y	-	-	-
		COMP-1	-	-	Y	Y	-	-	-
		COMP-2	-	-	-	Y	-	-	-

Formal parameter / Actual argument					Int64	UInt64	Float32	Float64	Char	Boolean	Decimal
		N(1) [DISPLAY]			-	-	-	-	Y	-	-
	Numeric	Binary	Integer	Unsigned	Y(*6)	Y(*6)	Y(*6)	Y(*6)	-	-	Y
				Signed	Y(*6)	-	Y(*6)	Y(*6)	-	-	Y
			Non Integer	Unsigned	-	-	Y(*6)	Y(*6)	-	-	Y
				Signed	-	-	Y(*6)	Y(*6)	-	-	Y
		Packed decimal	Integer	Unsigned	Y(*6)	Y(*6)	Y(*6)	Y(*6)	-	-	Y
				Signed	Y(*6)	-	Y(*6)	Y(*6)	-	-	Y
			Non Integer	Unsigned	-	-	Y(*6)	Y(*6)	-	-	Y
				Signed	-	-	Y(*6)	Y(*6)	-	-	Y
		Display	Integer	Unsigned	Y(*6)	Y(*6)	Y(*6)	Y(*6)	-	-	Y
				Signed	Y(*6)	-	Y(*6)	Y(*6)	-	-	Y
			Non Integer	Unsigned	-	-	Y(*6)	Y(*6)	-	-	Y
				Signed	-	-	Y(*6)	Y(*6)	-	-	Y
	Alphabetic data				-	-	-	-	-	-	-
	Alphanumeric character				-	-	-	-	-	-	-
	External floating-point				-	-	Y(*6)	Y(*6)	-	-	-
	National				-	-	-	-	-	-	-
	Boolean				-	-	-	-	-	Y(*11)	-
	Numeric edited				-	-	-	-	-	-	-
	Alphanumeric edited				-	-	-	-	-	-	-
	National edited				-	-	-	-	-	-	-
	Group (no type/weakly typed)				-	-	-	-	-	-	-
	Index				-	-	-	-	-	-	-
	Pointer data item				-	-	-	-	-	-	-

Table 12.17 Part (vi). Parameter Conformance - passing BY CONTENT.

Formal parameter / Actual argument		Class	Value (includes Enum type)	Object	String	Array type	COBOL type
COBOL type (*21)	Built-in CLR type	BINARY-CHAR UNSIGNED	-	-	Y	-	Y(*4)	-
		BINARY-SHORT SIGNED	-	-	Y	-	Y(*4)	-
		BINARY-LONG SIGNED	-	-	Y	-	Y(*4)	-
		BINARY-DOUBLE SIGNED	-	-	Y	-	Y(*4)	-
		S9(4) COMP-5	-	-	Y	-	Y(*4)	-
		S9(9) COMP-5	-	-	Y	-	Y(*4)	-

Formal parameter / Actual argument					Class	Value (includes Enum type)	Object	String	Array type	COBOL type
		S9(18) COMP-5			-	-	Y	-	Y(*4)	-
		COMP-1			-	-	Y	-	Y(*4)	-
		COMP-2			-	-	Y	-	Y(*4)	-
		N(1) [DISPLAY]			-	-	Y	-	Y(*4) (*22)	-
	Numeric	Binary	Integer	Unsigned	-	-	Y	-	-	Y(*7)
				Signed	-	-	Y	-	-	Y(*7)
			Non Integer	Unsigned	-	-	Y	-	-	Y(*7)
				Signed	-	-	Y	-	-	Y(*7)
		Packed decimal	Integer	Unsigned	-	-	Y	-	-	Y(*7)
				Signed	-	-	Y	-	-	Y(*7)
			Non Integer	Unsigned	-	-	Y	-	-	Y(*7)
				Signed	-	-	Y	-	-	Y(*7)
		Display	Integer	Unsigned	-	-	Y	-	-	Y(*7)
				Signed	-	-	Y	-	-	Y(*7)
			Non Integer	Unsigned	-	-	Y	-	-	Y(*7)
				Signed	-	-	Y	-	-	Y(*7)
	Alphabetic data				-	-	Y	Y(*8)	Y(*9)	Y(*7)
	Alphanumeric character				-	-	Y	Y(*8)	Y(*9)	Y(*7)
	External floating-point				-	-	Y	-	-	Y(*7)
	National				-	-	Y	Y(*8)	Y(*10)	Y(*7)
	Boolean				-	-	Y(*11)	-	-	Y(*7)
	Numeric edited				-	-	Y	-	-	Y(*7)
	Alphanumeric edited				-	-	Y	Y(*8)	-	Y(*7)
	National edited				-	-	Y	Y(*8)	-	Y(*7)
	Group (no type/weakly typed)				-	-	-	-	Y(*9)	Y(*7)
	Index				-	-	-	-	-	Y(*7)
	Pointer data item				-	-	-	-	-	Y(*7)

Table 12.18 Part (vii). Parameter Conformance - passing BY CONTENT.

Formal parameter / Actual argument		Int8	UInt8	Int16	UInt16	Int32	Uint32
Literal (*21)	Fixed-point numeric literal (positive integer)	-	-	Y(*12)	-	Y(*12)	-
	Fixed-point numeric literal (negative integer)	-	-	Y(*12)	-	Y(*12)	-
	Fixed-point numeric literal	-	-	-	-	-	-

Formal parameter / Actual argument		Int8	UInt8	Int16	UInt16	Int32	Uint32
	(noninteger)						
	Single-precision internal floating-point item	-	-	-	-	-	-
	Single-precision internal floating-point item	-	-	-	-	-	-
	Hexadecimal numeric literal	-	-	Y(*12)	-	Y(*12)	-
	Numeric literal	-	-	-	-	-	-
	Hexadecimal nonnumeric literal	-	-	-	-	-	-
	National language nonnumeric literal	-	-	-	-	-	-
	National hexadecimal nonnumeric literal	-	-	-	-	-	-
	Boolean literal	-	-	-	-	-	-
Figurative constant	ZERO	Y	Y	Y	Y	Y	Y
	SPACE	-	-	-	-	-	-
	NULL	-	-	-	-	-	-
	HIGH-VALUE	-	-	-	-	-	-
	LOW-VALUE	-	-	-	-	-	-
	QUOTE	-	-	-	-	-	-
	ALL literal	-	-	-	-	-	-
	Symbolic literal	-	-	-	-	-	-
ENUM field (*18)		Y(*19)	Y(*19)	Y(*19)	Y(*19)	Y(*19)	Y(*19)
TYPE OF operator (*18)		-	-	-	-	-	-

Table 12.19 Part (viii). Parameter Conformance - passing BY CONTENT.

Formal parameter / Actual argument		Int64	UInt64	Float32	Float64	Char	Boolean	Decimal
Literal (*21)	Fixed-point numeric literal (positive integer)	Y(*12)	-	Y(*12)	Y(*12)	-	-	-
	Fixed-point numeric literal (negative integer)	Y(*12)	-	Y(*12)	Y(*12)	-	-	-
	Fixed-point numeric literal (noninteger)	-	-	Y(*13)	Y(*13)	-	-	Y(*13)
	Single-precision internal floating-point item	-	-	Y(*14)	Y(*14)	-	-	-
	Single-precision internal floating-point item	-	-	Y(*14)	Y(*14)	-	-	-
	Hexadecimal numeric literal	Y(*12)	-	Y(*12)	Y(*12)	-	-	-
	Numeric literal	-	-	-	-	-	-	-
	Hexadecimal nonnumeric literal	-	-	-	-	-	-	-

Formal parameter / Actual argument		Int64	UInt64	Float32	Float64	Char	Boolean	Decimal
	National language nonnumeric literal	-	-	-	-	Y(*15)	-	-
	National hexadecimal nonnumeric literal	-	-	-	-	Y(*15)	-	-
	Boolean literal	-	-	-	-	-	Y(*11)	-
Figurative constant	ZERO	Y	Y	Y	Y	-	-	Y
	SPACE	-	-	-	-	Y	-	-
	NULL	-	-	-	-	-	-	-
	HIGH-VALUE	-	-	-	-	-	-	-
	LOW-VALUE	-	-	-	-	-	-	-
	QUOTE	-	-	-	-	-	-	-
	ALL literal	-	-	-	-	-	-	-
	Symbolic literal	-	-	-	-	-	-	-
ENUM field (*18)		-	Y(*19)	Y(*19)	Y(*19)	-	-	-
TYPE OF operator (*18)		-	-	-	-	-	-	-

Table 12.20 Part (ix). Parameter Conformance - passing BY CONTENT.

Formal parameter / Actual argument		Class	Value (includes Enum type)	Object	String	Array type	COBOL type
Literal (*21)	Fixed-point numeric literal (positive integer)	-	-	Y	-	-	Y(*7)
	Fixed-point numeric literal (negative integer)	-	-	Y	-	-	Y(*7)
	Fixed-point numeric literal (noninteger)	-	-	Y	-	-	Y(*7)
	Single-precision internal floating-point item	-	-	Y	-	-	Y(*7)
	Single-precision internal floating-point item	-	-	Y	-	-	Y(*7)
	Hexadecimal numeric literal	-	-	Y	-	-	Y(*7)
	Numeric literal	-	-	Y	Y(*8)	Y(*9)	Y(*7)
	Hexadecimal nonnumeric literal	-	-	Y	Y(*8)	Y(*9)	Y(*7)
	National language nonnumeric literal	-	-	Y	Y(*8)	Y(*9)	Y(*7)
	National hexadecimal nonnumeric literal	-	-	Y	Y(*8)	Y(*10)	Y(*7)

Formal parameter / Actual argument		Class	Value (includes Enum type)	Object	String	Array type	COBOL type
	Boolean literal	-	-	Y(*11)	-	-	Y(*7)
Figurative constant	ZERO	-	-	-	-	-	Y(*17)
	SPACE	-	-	-	-	-	Y(*17)
	NULL	Y	-	Y	Y	Y(*16)	Y(*17)
	HIGH-VALUE	-	-	-	-	-	Y(*17)
	LOW-VALUE	-	-	-	-	-	Y(*17)
	QUOTE	-	-	-	-	-	Y(*17)
	ALL literal	-	-	-	-	-	-
	Symbolic literal	-	-	-	-	-	-
ENUM field (*18)		-	Y(*3)	-	-	-	-
TYPE OF operator (*18)		Y(*20)	-	-	-	-	-

Notes on the Parameter Conformance BY CONTENT tables

*1 Must be an instance of the appropriate aliased type (System.xxxx)

*2 Must be an instance of the type to which the object refers or a type that is derived from it.

*3 Must be an instance of the type to which the object refers.

*4 The formal parameter must have the params specification, and be a one-dimensional array of the aliased type (System.xxxx) of the actual argument array.

*5 Conversion from the string type is enabled only for alphabet, alphanumeric, and national characters. The types of conversions performed depend on the runtime code set specified in the RCS compiler option:

RCS(UTF8-UCS2)

- When the receiving side is an alphanumeric item:
  Convert to UTF-8, then pad with spaces.
- When the receiving side is a national item:
  Perform space padding.

RCS(ACP-UCS2)

- When the receiving side is an alphanumeric item:
  Convert to ACP, then pad with spaces.
- When the receiving side is a national item:
  Perform space padding.

RCS(ACP)

- When the receiving side is an alphanumeric item:
  Convert to ACP, then pad with spaces.
- When the receiving side is a national item:
  Convert to ACP, then pad with spaces.

*6 Conversions to an integer are performed as implicit type conversions according to the receiving side descriptions. For overflow the results are rounded down. In the case of overloaded methods, the conversion will be to the narrowest matching formal parameter. See Determining Better Parameter Conformance rules 5 and 6.

*7 The actual argument and receiving side formal parameter must be a possible combination as shown in the table below (Y = possible, - = not possible).

Table 12.21 Possible combinations of arguments and parameters

Formal parameter / Actual argument	Group item	Bool	Integer signed	Integer Unsigned	Non-integer	Edited numeric	Alpha-numeric	Alphabet	National	Index	Pointer
Group item	Y	-	Y	Y	Y	Y	Y	Y	Y	-	-
Bool	Y	Y	-	-	-	-	Y	-	-	-	-
Integer signed	Y	-	Y	Y	Y	Y	Y	-	-	-	-
Integer unsigned	Y	-	Y	Y	Y	Y	Y	-	-	-	-
Non-integer	-	-	Y	Y	Y	Y	-	-	-	-	-
Edited numeric	Y	-	Y	Y	Y	Y	Y	-	-	-	-
Alpha-numeric character	Y	-	Y	Y	Y	Y	Y	Y	-	-	-
Alphabetic	Y	-	-	-	-	-	Y	Y	-	-	-
National	-	-	-	-	-	-	-	-	Y	-	-
Index	-	-	-	-	-	-	-	-	-	Y	-
Pointer	-	-	-	-	-	-	-	-	-	-	Y

*8 Character string conversion is performed as an implicit type conversion. The processing depends on the setting of the RCS compiler option. The following bullets outline the internal processing performed on input values:

- Alphabetic/alphanumeric item -> string
  RCS(UTF8-UCS2)/RCS(ACP-UCS2)/RCS(ACP): After removing trailing spaces, the string is converted to UCS-2
- National item -> string
  RCS(UTF8-UCS2)/RCS(ACP-UCS2): Trailing spaces are removed
  RCS(ACP): After removing trailing spaces, the string is converted to UCS-2
- Alphanumeric literal/hexadecimal-alphanumeric literal -> string
  RCS(UTF8-UCS2)/RCS(ACP-UCS2)/RCS(ACP): Characters are converted to UCS-2
- National literals/national hexadecimal literal -> string
  RCS(UTF8-UCS2)/RCS(ACP-UCS2):Data is just converted to a string
  RCS(ACP): Convert to UCS-2 before converting to string

*9 The actual parameter has to be a COBOL character type and the formal parameter must be a byte array or character array. The following implicit type conversions are performed:

- Alphabetic/alphanumeric/group item -> byte[]
  All of the area is converted into a byte array (including trailing spaces)
- Alphanumeric literals/hexadecimal-alphanumeric literals -> byte[]
  All of the literal is converted into a byte array (including trailing spaces)
- Alphabetic/alphanumeric item -> char[]
  UCS-2 strings are converted into a char array (including trailing spaces), and the entire area is passed in the call.
- Alphanumeric literals/hexadecimal-alphanumeric literal -> char[]
  UCS-2 strings are converted into a char array (including trailing spaces), and the results passed in the call.

*10 The following implicit type conversions are performed. The behavior depends on the setting of the RCS compiler option:

- National item -> char[]
  RCS(UTF8-UCS2)/RCS(ACP-UCS2): All of the area is converted into a char array (including trailing spaces), and the result passed in the call.
  RCS(ACP): The entire national item is converted to UCS-2 (including trailing spaces), converted to a char array, then the result passed in the call.

- National literal /national hexadecimal literal -> char[]
  RCS(UTF8-UCS2)/RCS(ACP-UCS2): All of the literal is converted into a char array (including trailing spaces), and the results passed in the call.
  RCS(ACP): All of the literal is converted to UCS-2 (including trailing spaces), then converted into a char array, and the result passed in the call.

*11 Only acceptable when the length is one. Implicit conversion into the System.Boolean type is performed.

*12 Integers are considered to be signed whether or not a sign is present. They are converted to a CLR type depending on the size of the value as shown in the following table:

Table 12.22 Integer Mapping to CLR Data Types

Value range	Mapped to
-32,768 to 32,767	int16
-2,147,483,648 to 2,147,483,647	int32
Outside these ranges	int64

When the formal parameter is a COBOL type, the argument is treated as a signed integer.

*13 Non-integer literals are mapped to non-integer native COBOL types.

*14 Floating-point literals are all mapped to float32.

*15 The literals are limited to a single character.

*16 Even if the formal parameter is a multi-dimensional array this is acceptable.

*17 The actual argument and receiving side formal parameter must be a possible combination as shown in the table below (Y = possible, x = not possible).

Table 12.23 Possible combinations of arguments and parameters

Formal parameter / Actual argument	Group item	Bool	Integer signed	Integer Unsigned	Non-integer	Edited numeric	Alpha-numeric	Alphabet	National	Index	Pointer
ZERO	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-
SPACE	Y	-	-	-	-	-	Y	Y	Y	-	-
NULL	-	-	-	-	-	-	-	-	-	-	Y
HIGH-VALUE	Y	-	-	-	-	-	Y	Y	Y	-	-
LOW-VALUE	Y	-	-	-	-	-	Y	Y	Y	-	-
QUOTE	Y	-	-	-	-	-	Y	Y	-	-	-

*18 Can be specified in either the USING or PROPERTY phrase of the CUSTOM-ATTRIBUTE clause in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION; but cannot be specified in an INVOKE statement or an inline invocation of a method.

*19 The conformance is dependent on the base type of the enumeration.

*20 The formal parameter must be a System.Type type.

*21 When the actual parameter is a COBOL character type, COBOL numeric type, numeric literal or nonnumeric literal, and, the corresponding formal parameter can be either a COBOL type or a CLR type, the COBOL type conforms before the CLR type.

*22 If the program is compiled with option RCS(ACP), converts to UCS-2, then passes it to the invocation destination.

Determining the Best Interface/Parameters (Overload Resolution)

When a method is invoked the compiler attempts to find the method that best matches the invocation signature (the combination of method name, number of parameters, parameter types, return type and visibility, that characterizes a method's interface). It is possible that more than one method may provide a match to the method invocation, so the compiler attempts to find which method has the best match by reviewing how actual arguments have been determined to map to the formal parameters in the receiving lists. For example, if the parameters of one candidate interface are all identical types to those of the invoking interface, while the parameters of another candidate interface all require some type conversions in order to conform, then the first interface is considered to be a better match than the second. (Conversions apply to arguments passed BY VALUE or BY CONTENT - parameters passed BY REFERENCE need to have identical sending and receiving types.)

If the compiler cannot determine that one interface is the best from the set of candidate interfaces, then the invocation is considered to be ambiguous and a compiler error is issued.

This topic describes how the compiler determines which interface is the best. This process is often referred to as "overload resolution" - i.e. the process of resolving which method to use when the method name is overloaded.

Candidate Methods

First the compiler determines the set of candidate methods.

If the set contains only one member, then that method is the best match.

Definition of Best Interface

Given an argument list S with a set of argument types {S1, S2, ..., SN} and two applicable methods MX and MY with formal parameter types {X1, X2, ..., XN} and {Y1, Y2, ..., YN}, MX is defined to be a better method than MY if

- for each parameter, the conformance of Sn to Xn is not worse than the conformance of Sn to Yn, and
- for at least one parameter, the conformance of Sn to Xn is better than the conformance of Sn to Yn.

From this definition, you can see that if one interface is better than another interface in some parameters, but is worse in others, then neither interface will be considered to be the better interface.

Determining Better Parameter Conformance

Suppose the type, S1, of a sending (also called "actual") argument conforms to the types X1 and Y1, of two receiving (also called "formal" or "dummy") parameters, the better conformance is determined by the following rules:

1. If X1 and Y1 are the same then the sending parameter is considered to conform to neither receiving parameter better than the other.
2. If either of X1 or Y1 is a native COBOL data type, and the other is a CLR type, then S1 is considered to conform better to the native COBOL type.
3. When S1 and X1 are the same type, and are different from Y1, S1 conforms better to X1 than Y1.
4. When S1 and Y1 are the same type, and are different from X1, S1 conforms better to Y1 than X1.
5. When X1 conforms to Y1 and Y1 does not conform to X1, S1 conforms better to X1 than Y1.
6. When Y1 conforms to X1 and X1 does not conform to Y1, S1 conforms better to Y1 than X1.
7. In other cases, it cannot be determined which of X1 and Y1 provide the better conformance to S1.

Treatment of Return Values

Values returned in RETURNING items are either made to conform by following the standard rules for the MOVE statement or, for object references, the rules of the SET statement. For these items the RETURNING item in the called program or method is regarded as the sending item, and the RETURNING item in the calling/invoking code is regarded as the receiving item.

12.8.2.4 Exception Conditions

Exception conditions generate an exception object. An exception object is an object instance of a class that inherits from the System.Exception class.

12.8.2.4.1 Exception Object

With the exception of the TRY statement, the generation of an exception object causes one of the two events below. For details on the TRY statement, refer to TRY in the Statements section below.

1. The DECLARATIVES section, within the currently executing program, that declared the class of the exception object in its USE statement, is executed. The code in the DECLARATIVES section is executed and, when it processes successfully (i.e. there is no unit of code, no program is called, nor method invoked, that terminates with an EXIT PROGRAM, or STOP RUN statement), an implicit EXIT PROGRAM or EXIT METHOD statement is executed.
2. When there is no USE statement in the current source unit that declares the class of the exception object, and this exception object inherits the System.Exception class, an implicit EXIT PROGRAM or EXIT METHOD statement with the RAISING clause, for which the exception object is specified, is executed.

12.8.2.5 ON SIZE ERROR Phrase

When a resultant identifier is defined with USAGE BINARY-CHAR, BINARY-SHORT, BINARY-LONG or BINARY-DOUBLE, the fourth condition of "Conditions Where a Size Error Condition Occurs" in the "ON SIZE ERROR" section of "Chapter 6" changes to the following:

- The absolute value obtained by an arithmetic operation is not in the range shown in Table 12.1 "Amount of Memory Allocated and Range of Values for BINARY-CHAR/SHORT/LONG/DOUBLE Data Items".

12.8.3 Statements

12.8.3.1 Conditional statements and imperative statements

The following .NET programming specific statements are imperative statements:

- TRY
- INVOKE
- RAISE
- RESUME
- USE
- EXIT TRY
- EXIT METHOD
- EXIT PROGRAM

For more details about conditional and imperative statements, refer to "Composing Procedure Divisions".

12.8.3.2 ALTER

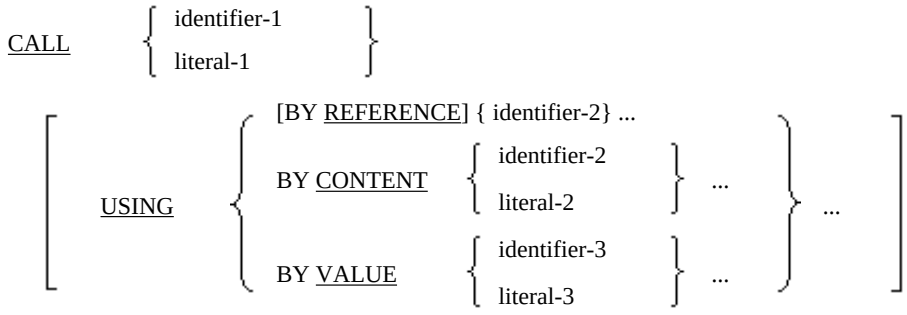
Syntax Rules

The ALTER statement is not supported in the NetCOBOL for .NET compiler.

12.8.3.3 CALL

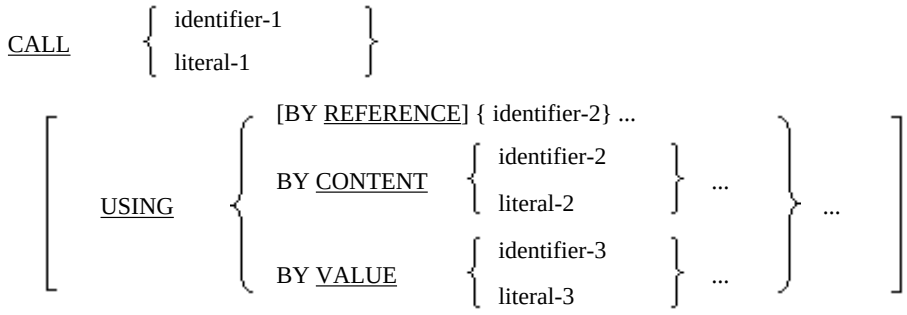
Format 1

(ON OVERFLOW)



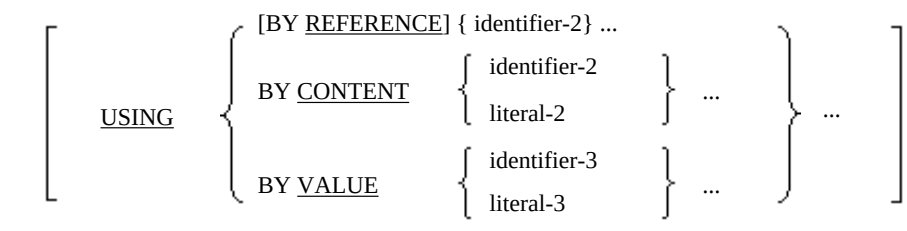
Format 2

(ON EXCEPTION)



Format 3

(Program Prototype)



Syntax Rules

- Rules for all Formats

  Identifier-4 must be a data item that has been defined in the FILE SECTION, WORKING-STORAGE SECTION, LINKAGE SECTION or LOCAL-STORAGE SECTION.

- Rules for Formats 1 and 2

  1. Identifier-2 and identifier-3 must be data items that have been defined in the BASED-STORAGE, FILE, WORKING-STORAGE, CONSTANT, LINKAGE or LOCAL-STORAGE section.
  2. Identifier-3 must be one of the following
     - An integer data item, of nine digits or less, of USAGE COMPUTATIONAL-5
     - A data item of USAGE BINARY-SHORT SIGNED
     - A data item of USAGE BINARY-LONG SIGNED
  3. Literal-3 must be an integer of nine digits or less
  4. For other syntax rules, see the topic titled "CALL Statement (Inter-program Communication)" in the "Statements" section of Chapter 6, "Procedure Division".

- Rules for Format 3

  1. Identifier-2 must be a data item that has been defined in the BASED-STORAGE, FILE, WORKING-STORAGE, LOCAL-STORAGE or LINKAGE section.
  2. Identifier-3 must be a data item defined in the BASED-STORAGE, FILE, WORKING-STORAGE, LOCAL-STORAGE, CONSTANT or LINKAGE section.
  3. The USING clause follows the rules of the USING clause of the program prototype's PROCEDURE DIVISION header.
  4. When either BY CONTENT or BY REFERENCE should be specified for an argument, the corresponding program-prototype PROCEDURE DIVISION header should specify BY REFERENCE in the corresponding parameters.
  5. Literal-2 must conform to the corresponding formal parameter as specified for the conformance of elementary items that are passed BY CONTENT. See page.
  6. Literal-2 must be a numeric literal, a non-numeric literal, a national language non-numeric literal, or a Boolean literal.

General Rules

- Rules for all formats

  Identifier-1, identifier-2, identifier-3, and identifier-4 are evaluated at the beginning of the execution of the CALL statement.

- Rules for Formats 1 and 2

  1. When the RETURNING phrase is written, the result of the program identified by identifier-1 or literal-1 is stored in identifier-4. However, when control is returned from the called program with the notification of an exception, identifer-4 does not reflect the result of the program.
  2. For other general rules, see the topic titled "CALL Statement (Inter-program Communication)" in the "Statements" section of Chapter 6, "Procedure Division".

- Rules for Format 3

  1. When BY VALUE is specified, the value of the parameter is passed directly to the called program.
     If the called program changes the value that is passed to it, the value of identifier-2 in the calling program is not changed.

2. When none of BY REFERENCE, BY CONTENT or BY VALUE is specified for the parameter, passing the parameter is decided as follows:

   - When the corresponding formal parameter is BY REFERENCE, BY REFERENCE is assumed. However, if a literal, EXCEPTION-OBJECT, NULL, SELF or data item defined in the CONSTANT section is specified, BY CONTENT is assumed.
   - When the corresponding formal parameter is BY VALUE, BY VALUE is assumed.

3. Program prototype-name-1 is used to determine which program to call.
4. Control is passed to the invoked program in accordance with the program invocation rules.
5. When the RETURNING phrase is written, the result of the program identified by prototype-name-1 is stored in identifier_4. However, when control is returned from the called program with the notification of an exception, identifer-4 does not reflect the result of the program.
6. Do not specify the RETURNING phrase when the called program does not return the result of the program in a RETURNING item.

12.8.3.4 DISPLAY

General Rules

If a BINARY-CHAR/SHORT/LONG/DOUBLE data item is specified for identifier-1, it is treated like a binary data item with the USAGE COMP-5 clause, and displayed in the following format:

Table 12.24 DISPLAY Representation of BINARY-CHAR/SHORT/LONG/DOUBLE Data Items

USAGE Clause	Maximum Value	Representation (*1)
BINARY-CHAR UNSIGNED	+255	9(3)
BINARY-SHORT SIGNED	+32,767	s9(5)
BINARY-LONG SIGNED	+2,147,483,647	s9(10)
BINARY-DOUBLE SIGNED	+9,223,372,036,854,775,807	s9(19)

*1: "s"" indicates a sign (+ or -), "9(n)" indicates the equivalent display numeric item used to display the number.

For other rules, see the topic titled "DISPLAY Statement (Nucleus)" in the "Statements" section of Chapter 6, "Procedure Division".

12.8.3.5 ENTRY

Syntax Rules

The NetCOBOL for .NET compiler does not support the ENTRY statement.

12.8.3.6 EXIT

The EXIT PROGRAM statement indicates the logical end of a called program.

The EXIT METHOD statement indicates the logical end of an invoked method. The EXIT TRY statement indicates the logical end of the TRY block and the CATCH block.

Format 1

(Program)

```
EXIT PROGRAM
    [RAISING identifier-1]
```

Format 2

(Method)

EXIT METHOD

[RAISING identifier-1]

Format 3

(TRY block/CATCH block)

EXIT TRY

Syntax Rules

- Rule for Formats 1 and 2

  Identifier-1 must be the name of an object reference that references a class that is specified in the RAISING clause of the PROCEDURE DIVISION header of the source element that includes the EXIT statement. The class must inherit from the System.Exception class.

- Rule for Format 1

  Do not write the EXIT PROGRAM statement in a method definition.

- Rule for Format 2

  The EXIT METHOD statement can only be specified in the PROCEDURE DIVISION of a method.

- Rule for Format 3

  The EXIT TRY statement can only be specified for the TRY statement. However, it cannot be specified for the FINALLY block of the innermost TRY statement that includes the EXIT TRY statement.

General Rules

- Rule for Formats 1 and 2

  When the RAISING phrase is specified and an exception condition occurs, the object referenced by identifier-1 becomes the exception object.

- Rule for Format 1

  When a program has not been called and the EXIT PROGRAM statement is executed, EXIT PROGRAM is treated like a CONTINUE statement. No exception condition is generated, even if the RAISING phrase is specified.

- Rule for Format 2

  Executing the EXIT METHOD statement terminates the method. Control returns to the invoking statement. If the RETURNING phrase has been specified in the PROCEDURE DIVISION header of the method that includes the EXIT METHOD statement, the data value referenced in the RETURNING phrase is passed to the invoking program.

- Rules for Format 3

  1. The EXIT TRY sentence is associated with the innermost TRY statement in which it is included. The associated TRY statement ends if the EXIT TRY statement is executed. The order of execution is as follows:
  2. The FINALLY block is executed if it is specified in the associated TRY statement.
  3. Control is moved to just behind END-TRY of the associated TRY statement.

12.8.3.7 GO TO

Syntax Rules

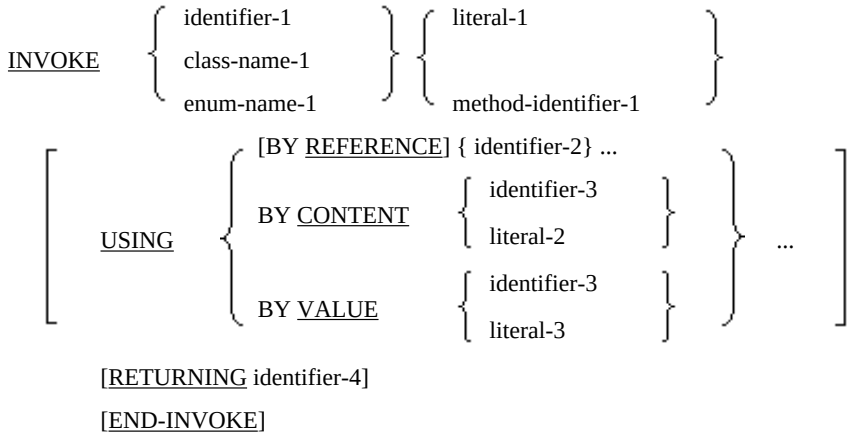
The NetCOBOL for .NET compiler does not support a format 1 GO TO statement with procedure-name-1 omitted.

12.8.3.8 INVOKE

The invoke statement is used for invoking methods.

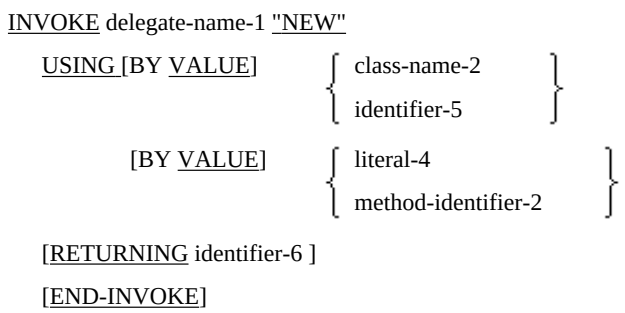
Format 1

(Usual method call)



Format 2

(Delegate constructor)



Syntax Rules

- Rules for all Formats

  Method name NEW is a compiler reserved name that indicates a constructor.

- Rules for Format 1

  1. Class-name-1, enum-name-1, and method-identifier-1 should each be specified by a class-specifier, enum-specifier, or method-specifier in the REPOSITORY paragraph.
  2. Identifier-1 should be an object identifier, other than NULL.
  3. Literal-1 is a nonnumeric or national language non-numeric literal. It should not be a hexadecimal-non-numeric literal or a national hexadecimal non-numeric literal.
  4. When NEW is specified for literal-1, you should specify either class-name-1 or SUPER for identifier-1. However, when SUPER is specified for identifier-1, the INVOKE statement should only be specified in the procedure division of a "NEW" method.
  5. Identifier-2 must be a data name defined in the BASED-STORAGE, FILE, WORKING-STORAGE, LINKAGE or LOCAL-STORAGE section.

6. Identifier-3 must be a data name defined as SELF, NULL or the EXCEPTION-OBJECT or a data item defined in the BASED-STORAGE, FILE, WORKING-STORAGE, CONSTANT, LINKAGE or LOCAL-STORAGE section.
7. Indentifier-4 must be a data name defined in the FILE, WORKING-STORAGE, LINKAGE or LOCAL-STORAGE section.
8. When the PROCEDURE DIVISION header of the method being invoked includes a USING clause, the INVOKE statement must also have a USING clause. If the PROCEDURE DIVISION header of the method being invoked does not have a USING clause, do not include a USING clause in the INVOKE statement.
9. When the PROCEDURE DIVISION header of the method being invoked has a RETURNING clause, the INVOKE statement does NOT have to have a RETURNING clause. However, if the PROCEDURE DIVISION header of the method being invoked does not have a RETURNING clause a RETURNING clause must not be code in the INVOKE statement.
   This rule does not apply to invoking a constructor (NEW).
10. Identifier-2, identifier-3, identifier-4, literal-2, and literal-3 should follow the rules for conformance of parameters as described.
11. When BY REFERENCE is associated, explicitly or implicitly, with identifier-2, reference modification cannot be used.
12. When BY REFERENCE is associated, explicitly or implicitly, with identifier-2, and identifier-2 is an internal Boolean item, the internal Boolean item should be defined to start on a byte boundary.
13. Identifier-4 cannot be reference modified.
14. If identfier-4 is an internal Boolean item, identifier-4 must be defined to start on a byte boundary.
15. When the BY VALUE phrase is defined for an argument, the corresponding parameter in the PROCEDURE DIVISION header of the method to be invoked must also have the BY VALUE phrase.
16. The method being invoked should be determined uniquely by the combination of class-name-1, enum-name-1 or identifier-1, method name and the actual arguments specified in the USING phrase. When there are two or more methods that can be invoked, and one of them satisfies the following conditions, the method being invoked can be determined uniquely.
    a. One or more formal parameters better conform to the actual argument
    b. And, for the arguments other than above, no other method conforms better than this method.

- Rules for Format 2

1. Delegate-name-1 should be a delegate-name declared in the REPOSITORY paragraph of the class that contains this source element.
2. Class-name-2 should be a class-name declared in the REPOSITORY paragraph of the class that contains this source element.
3. Identifier-5 should be an object reference item or the predefined object identifier SELF.
4. Literal-4 should be an alphanumeric literal. However, it cannot be a hexadecimal-alphanumeric literal, and its value cannot be "NEW".
5. Method-identifier-2 must be the method identifier declared in the REPOSITORY paragraph of the class that contains this source element.
6. Identifier-6 should be an object reference item that can contain the object reference of delegate-name-1.

General Rules

- Rules for Format 1

1. Literal-1 identifies the name of the method to be called. When the method being invoked is a COBOL method, the content of literal-1 must be the external name of the called method.

2. Identifier-1 identifies the object instance. The method is identified by selecting from the methods associated with the type of the object specified by identifier-1 and subject to the following rules:

    a. If Literal-1 is specified, methods with the name Literal-1 are identified regardless of whether those methods have type parameters.

    b. If method-identifier-1 is specified, the only methods identified are those that match the method-specifier definition of method-identifier-1 in the repository section. To match, the method must have the same name and the same number of type parameters as were specified in the method-specifier definition.

3. ENUM-name-1 identifies the ENUM in which to invoke a method. Literal-1 defines a static method in ENUM-name-1 or a parent class.

4. Class-name-1 identifies the class in which a method should be invoked. Literal-1 or method-identifier-1 identifies the method to be invoked as follows:

    a. When literal-1 is "NEW", the constructor of object instances for class-name-1 is identified.

    b. When literal-1 is not "NEW", it defines the corresponding static method in class-name-1 or a parent class, regardless of any type parameter specified.

    c. If method-identifier-1 is specified, the only methods identified are those that match the method-specifier definition of method-identifier-1 in the repository section. To match, the method must have the same name and the same number of type parameters as were specified in the method-specifier definition.

5. Identifier-3, identifier-4, literal-2 and literal-3 are actual parameters of this method call. Actual parameters are associated with the formal parameters specified in the USING specification of the PROCEDURE DIVISION header in the order they are listed. The method is then invoked matching the actual and formal parameters according to the rules of parameter conformance. If a called method has any type parameters, the actual type arguments must meet the constraints of the corresponding type parameter and are determined by the following rules:.

    - If method-identifier-1 is specified the actual type arguments are determined by the type arguments supplied in the definition of the method specifier with the name method-identifier-1.

    - If literal-1 is specified, the actual type arguments used are inferred using the following rules:

      (a). For each formal parameter of the method whose type is a type parameter, the actual type argument is inferred to be the type of the corresponding actual argument. For the purposes of inference, literals match CLR data types according to the following rules:

      - Integer literals are treated according to their size as shown in Table 12.10

      - Floating-point literals are treated as being of type System.Single

      - Alphanumeric literals are treated as being of type System.String

      - National literals with only a single character are treated as being of type System.Char and with more than a single character are treated as being of type System.String

      - Boolean literals with a length of one are treated as being of type System.Boolean

      (b).Type arguments that are inferred may only be of CLR data types

      (c). In order to be inferred, each type parameter must appear at least once in the formal parameter list (not including the return type)

6. When the RETURNING phrase is written, the result of the method identified by identifier-1 and literal-1 is stored in identifier-4. However, when control is returned from the called program with the notification of an exception, identifer-4 does not reflect the result of the program.

7. When the INVOKE statement is executed, the method specified in the statement becomes active and control is passed to it. When the method exits normally control returns to the end of the INVOKE statement.

8. If the specified method cannot be invoked an exception is generated in the current execution unit.

9. When none of BY CONTENT, BY REFERENCE and BY VALUE are specified for any argument in the INVOKE statement, the system uses the corresponding specifications of the formal parameters defined in the PROCEDURE

DIVISION header of the method being invoked. However, when EXCEPTION-OBJECT, NULL, SELF or a class name is specified as an argument, BY CONTENT is assumed.

10. When the INVOKE sentence is executed, the values of the arguments specified in the USING clause are made available to the called method according to rules of 12.8.2.3.1 Parameter Conformance.
11. INVOKE statements can be included in methods that are themselves invoked. It is also possible for methods to invoke themselves directly or indirectly.
12. END-INVOKE marks the end of the INVOKE statement.

- Rules for Format 2

1. Format 2 is a delegate constructor.
2. Identifier-name-5 identifies the object instance. A method within an object instance identified by the object reference is associated with the generated delegate object. The method is identified as follows by the specification of literal-4 or method-identifier-2:
   a. If literal-4 is specified, the object method specified by literal-4 is identified, regardless of any type parameters specified.
   b. If method-identifier-2 is specified, the only methods identified are those that match the method-specifier definition of method-identifier-2 in the repository section. To match, the method must have the same name and the same number of type parameters as were specified in the method-specifier definition.
3. Class-name-2 identifies the class. A class identified by this class name or a static method of the parent class is associated with the generated delegate object. At this time, the method is identified as follows by the specification of literal-4 or method-identifier-2:
   a. If literal-4 is specified, the same static method as the name defined in the class specified by class-name-2 or the parent class is identified regardless of any type parameter specified.
   b. If method-identifier-2 is specified, the only methods identified are those with the same number of type parameters as the type argument described in the USING specification of the method-specifier that defines method-identifier-2 in the static method defined in the class specified by the class-identifier-2 or the parent class.
4. The signature of the method defined in the INVOKE delegate-name "NEW" statement, should match the signature of delegate-name.
5. A delegate object is returned in identifier-6. This object can be used to invoke indirectly the method specified in the delegate constructor INVOKE statement. To do this you invoke the "Invoke" method of the object referenced by identifier-6. See the NetCOBOL for .NET User Guide for an example of using delegates.

12.8.3.9 PERFORM

The PERFORM statement is used to repeat a specified group of statements for each element in a data collection.

Format 6

```
PERFORM VARYING identifier-1
        { THRU     }
        { THROUGH  }  identifier-2
imperative-statement-1
END-PERFORM
```

Syntax Rules

- Rules for Format 6

1. THRU and THROUGH are synonymous.
2. Identifier-1 must be a data item.

3. Identifier-2 must be an object-identifier that identifies a data collection. However, it must not be a predefined object identifier.
4. The type specified for identifier-1 must be one that can have elements from the data collection specified for identifier-2 assigned to it.
5. The type of the object identifier specified for identifier-2 must be one of the following:
    a. A type that implements the System.Collections.IEnumerable interface.
    b. A type that implements the System.Collections.Generic.IEnumerable interface.
    c. A type that satisfies the following conditions:

    - The type has an accessible instance method named GetEnumerator. The GetEnumerator method does not have arguments and returns a type T.

    - The type T that contains the accessible instance method named MoveNext. The MoveNext method does not have arguments and returns a Boolean type.

    - The type T contains the accessible instance property named Current. The Current property has a get accessor.

General Rules

- Rules for Format 6

1. he END-PERFORM phrase terminates the range of the in-line PERFORM statement.
2. The in-line PERFORM statement repeatedly executes for all elements in the data collection specified in identifier-2.
3. If the PERFORM statement is executed, the first element in the data collection specified in identifier-2 is stored in identifier-1 before executing the first statement in the specified group of statements.
   After the last statement in the specified group of statements has been executed, the next element in the data collection specified in identifier-2 is checked.
   If the element exists, it is stored in identifier-1 and the specified group of statements is executed again. Otherwise, control returns to the end of the PERFORM statement.
4. The identifier-1 must not be changed while executing the group of statements to preserve the order in which items from the data collection are enumerated.
5. If the EXIT statement with TEST is specified in imperative-statement-1, the element in the data collection specified in identifier-2 is checked.

12.8.3.10 RAISE

This section describes how the RAISE statement is used in .NET programming.

Format

RAISE identifier-1

Syntax Rules

1. Identifier-1 must be an object reference identifier. It must also be a class that inherits System.Exception class.
2. A RAISE statement that omits identifier-1 can only be specified in the CATCH block of the innermost TRY statement that includes the RAISE statement.

General Rules

1. If identifier-1 has not been specified, the RAISE statement is associated with the innermost TRY statement that includes the RAISE statement.

2. If a RAISE statement with identifier-1 specified is executed, the exception object specified by identifier-1 is generated. Otherwise, the procedures are executed in the following order:
    a. If FINALLY is specified in the TRY statement, the FINALLY block is executed.
    b. When control returns to CATCH block in a TRY statement that includes a RAISE statement, the outside of TRY statement is notified of the exception object.
3. If there is no applicable declarative procedure, even if identifier-1 has been specified, the RAISE statement will have the same effect as the CONTINUE statement unless it is in the TRY statement.
4. If a RAISE statement with identifier-1 described in the TRY statement is executed, a System.NullReference exception occurs if identifier-1 has NULL object.

12.8.3.11 RESUME

The RESUME statement transfers control to a specified procedure-name.

Format

RESUME AT procedure-name-1

Syntax Rules

1. The RESUME statement may be specified only in a USE procedure of an exception-object.
2. Procedure-name-1 is a procedure-name in the non-declarative part of the method or program.

General Rules

When the RESUME statement is executed, control is transferred to procedure-name-1 as if a GO TO procedure-name-1 were executed. Execution of the exception declarative that the RESUME statement describes is terminated.

12.8.3.12 SET

The SET statement can be used to transcribe object reference values.

Format 1

(Setting an object reference to an object reference)

SET { identifier-1} ... TO identifier-2

Format 2

(Setting data types other than object references from an object reference)

SET { identifier-3} ... TO identifier-4

Format 3

(Setting an object reference from data types other than object references)

SET { identifier-5} ... TO { identifier-6 | literal-1 }

Format 4

(Setting a byte array object as the storage area of the record in the LINKAGE SECTION)

SET ADDRESS OF record-1 TO identifier-7

Syntax Rules

- Rules for Format 1 (Setting an object reference to an object reference)

  1. Identifier-1 and identifier-2 should be object reference identifiers.
  2. The table below shows the combinations of identifier-1 and identifier-2 permitted in format 1 of the SET statement.

Table 12.25 Assignment compatibility for Format 1 of the SET statement.

Sending side (identifier-2)		Receiving side (identifier-1)				
		Class name		Interface name	Delegate name	Enum name
		Reference Type	Value Type			
Predefined object identifier NULL		Y		Y	Y	
Predefined object identifier SELF		Y *4		Y *5		
Class name	Reference Type	Y *1	Y *1	Y *2		
	Value Type	Y *1	Y *1	Y *2		
Interface name		Y *8		Y *3		
Delegate name		Y *1			Y *6	
Enum name		Y *1				Y *7

*1 If the class specified for the receiving side object is not inherited, none of the class, delegate or enumerator specified for the sending side object is transferred.

*2 If the interface specified for the receiving side object is not implemented, the class specified for the sending side object is not transferred.

*3 If the interface specified for the receiving side object is not inherited, the interface specified for the sending side object is not transferred.

*4 The statement should appear in an object method of a class that conforms to the receiving object's class.

*5 The statement must appear in an object method of a class that conforms to the receiving object's class.

*6 The delegate name on the receiving side must be the same as that on the sending side.

*7 The ENUM name on the receiving side must be the same as that on the sending side.

*8 The class name of the receiving side must be System.Object.

- Rules for Formats 2 and 3

  1. Identifier-4 and identifier-5 should be an object reference identifier of one of the following objects:

System.Byte	System.SByte
System.Int16	System.UInt16
System.Int32	System.UInt32
System.Int64	System.UInt64
System.Single	System.Double
System.Decimal	System.Char
System.Boolean	System.String
System.Byte[]	System.Char[]
System.Object	ENUM

2. When conversion from UCS-2 to ACP is indicated in Tables 12.14 and 12.15 and associated notes, it is important to note that:
   * If a UCS-2 character value cannot be represented in ACP, an execution time exception will occur.

- Rules for Format 2 (Setting data types other than object references from an object reference)

1. Identifier-3 can be a numeric item, an alphanumeric item, national item, a floating-point item, or a Boolean item.
2. The table below shows the combinations of identifier-3 and identifier-4 permitted in format 2 of the SET statement.

Table 12.26 Part (i). Assignment compatibility for Format 2 of the SET statement.

Sending side (identifier-4)	Receiving item (identifier-3)							
					Numeric item			
					Unsigned		Signed	
	BINARY-CHAR UNSIGNED	BINARY-SHORT SIGNED	BINARY-LONG SIGNED	BINARY-DOUBLE SIGNED	Integer	Non-integer	Integer	Non-integer
Byte	Y	Y	Y	Y	Y *1	Y *1	Y *1	Y *1
SByte	Y *6	Y	Y	Y			Y *1	Y *1
Int16	Y *6	Y	Y	Y			Y *1	Y *1
UInt16	Y *6	Y *6	Y	Y	Y *1	Y *1	Y *1	Y *1
Int32	Y *6	Y *6	Y	Y			Y *1	Y *1
UInt32	Y *6	Y *6	Y *6	Y	Y *1	Y *1	Y *1	Y *1
Int64	Y *6	Y *6	Y *6	Y			Y *1	Y *1
UInt64	Y *6	Y *6	Y *6	Y *6	Y *1	Y *1	Y *1	Y *1
Single		Y *10	Y *10	Y *10			Y *10	Y *10
Double		Y *10	Y *10	Y *10			Y *10	Y *10
Decimal		Y *2	Y *2	Y *2			Y *2	Y *2
Char								
Boolean								
String								
Byte[]								
Char[]								
Enum	Y *7	Y *7	Y *7	Y *7			Y *1 *7	

Table 12.27 Part (ii). Assignment compatibility for Format 2 of the SET statement.

Sending side (identifier-4)	Receiving item (identifier-3)							
	Alphabetic Alpha-numeric character item	National item	Group item (non-typed item, weakly typed)	External floating-point item	Internal floating-point		Boolean item	
					Single precision	Double precision	Internal	External
Byte								
SByte								
Int16								
UInt16								
Int32								
UInt32								
Int64								
UInt64								
Single				Y *8	Y *8	Y *8		
Double				Y *8		Y *8		
Decimal								
Char		Y *3						
Boolean							Y *3	Y *3
String	Y *4 *5	Y *5 *9						
Byte[]	Y		Y					
Char[]		Y *9						
Enum								

*1 It is possible to substitute if there are sufficient digits in the receiving item to store the value being sent.

*2 A CLR exception is generated when there are too few digits.

*3 The length of the item on the receiving side can only be 1. When the sending side is a char type and the COBOL program is compiled with RCS(ACP), the UCS-2 value of the sending char is converted to the ACP of the receiving item.

*4 Strings are converted from UCS-2 to UTF-8. If the program is compiled with RCS(ACP) or RCS(ACP-UCS2) the strings are converted from UCS-2 to ACP.

*5 The receiving item is space filled from the point the sending characters terminate to the end of the receiving item. When the sending side is NULL, spaces are substituted.

*6 When the sending item is bigger than the receiving item only the portion of the sending item that fits in the receiving item is transcribed.

*7 The rules for the substitution depend on the base type of the enumeration.

*8 When the sending side is single, it is treated as USAGE COMP-1.
When the sending side is double, it is treated as USAGE COMP-2.
The rules for the MOVE statement are applied to the substitution.

*9 If the program is compiled with RCS(ACP), the string data is converted from UCS-2 to ACP in the receiving item.

*10 Follows the rules specified in "Rules for Moving Data" in Chapter 6, with the sending item treated as a floating-point item.

- Rules for Format 3 (Setting an object reference from data types other than object references)

1. Identifier-6 should be a numeric item, an alphanumeric item, national item, a floating-point item, a Boolean item, an edited numeric item, an alphanumeric edited item, or a national edited item.
2. The following table shows the combinations of identifier-5, identifier-6, and literal-1 that are permitted in the format 3 of the SET statement.

Table 12.28 Part (i). Assignment compatibility for Format 3 of the SET statement.

Receiving side / Sending side			SByte	Byte	Int16	UInt16	Int32	Uint32
BINARY-CHAR UNSIGNED			Y(*12)	Y	Y	Y	Y	Y
BINARY-SHORT SIGNED			Y(*12)	Y(*12)	Y	Y(*11)	Y	Y(*11)
BINARY-LONG SIGNED			Y(*12)	Y(*12)	Y(*12)	Y(*12)	Y	Y(*11)
BINARY-DOUBLE SIGNED			Y(*12)	Y(*12)	Y(*12)	Y(*12)	Y(*12)	Y(*12)
Numeric item	**Unsigned**	**Integer**	Y	Y	Y	Y	Y	Y
		Non-integer						
	Signed	**Integer**	Y		Y		Y	
		Non-integer						
Alphabetic data item								
Alphanumeric character item/ Alphanumeric character function								
External floating-point item								
Internal floating-point item	**Single precision**							
	Double precision							
National item/ function								
Internal Boolean item								
External Boolean item								
Edited numeric item								
Alphanumeric edited item								
National edited item								
Group item (no type or weakly typed)								
Index data item								
Pointer data item								
Fixed-point literal	**Unsigned**	**Integer**	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)
		Non-integer						
	Signed	**Integer**	Y(*1)		Y(*1)		Y(*1)	

Receiving side / Sending side			SByte	Byte	Int16	UInt16	Int32	Uint32
		Non-integer						
Hexadecimal numeric literal			Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)	Y(*1)
Floating-point literal								
Nonnumeric literal **Hexadecimal nonnumeric literal**								
National hexadecimal literal								
Boolean literal								
ZERO			Y	Y	Y	Y	Y	Y
SPACE								
HIGH-VALUE, LOW-VALUE								
QUOTE, symbolic-character								

Table 12.29 Part (ii). Assignment compatibility for Format 3 of the SET statement.

Receiving side / Sending side			Int64	UInt64	Single	Double	Decimal	Char
BINARY-CHAR UNSIGNED			Y	Y	Y(*14)	Y(*14)	Y	
BINARY-SHORT SIGNED			Y	Y(*11)	Y(*14)	Y(*14)	Y	
BINARY-LONG SIGNED			Y	Y(*11)	Y(*14)	Y(*14)	Y	
BINARY-DOUBLE SIGNED			Y	Y(*11)	Y(*14)	Y(*14)	Y	
Numeric item	**Unsigned**	**Integer**	Y	Y	Y(*14)	Y(*14)	Y	
		Non-integer			Y(*14)	Y(*14)	Y	
	Signed	**Integer**	Y		Y(*14)	Y(*14)	Y	
		Non-integer			Y(*14)	Y(*14)	Y	
Alphabetic data item								
Alphanumeric character item/ **Alphanumeric character function**								
External floating-point item					Y(*14)	Y(*14)		
Internal floating-point item	**Single precision**				Y(*14)	Y(*14)		
	Double precision					Y(*14)		
National item/ function								Y(*2) (*15)
Internal Boolean item								
External Boolean item								
Edited numeric item								

Receiving side / Sending side			Int64	UInt64	Single	Double	Decimal	Char
Alphanumeric edited item								
National edited item								
Group item (no type or weakly typed)								
Index data item								
Pointer data item								
Fixed-point literal	**Unsigned**	**Integer**	Y(*1)	Y(*1)	Y(*14)	Y(*14)	Y(*13)	
		Non-integer			Y(*14)	Y(*14)	Y(*13)	
	Signed	**Integer**	Y(*1)		Y(*14)	Y(*14)	Y(*13)	
		Non-integer			Y(*14)	Y(*14)	Y(*13)	
Hexadecimal numeric literal			Y(*1)	Y(*1)	Y(*14)	Y(*14)	Y(*13)	
Floating-point literal					Y(*14)	Y(*14)		
Nonnumeric literal **Hexadecimal nonnumeric literal**								
National hexadecimal literal								Y(*2) (*15)
Boolean literal								
ZERO			Y	Y	Y	Y	Y	
SPACE								Y
HIGH-VALUE, LOW-VALUE								
QUOTE, symbolic-character								

Table 12.30 Part (iii). Assignment compatibility for Format 3 of the SET statement.

Receiving side / Sending side			Boolean	String	Byte[]	Char[]	ENUM	Object
BINARY-CHAR UNSIGNED								Y(*5)
BINARY-SHORT SIGNED								Y(*5)
BINARY-LONG SIGNED								Y(*5)
BINARY-DOUBLE SIGNED								Y(*5)
Numeric item	**Unsigned**	**Integer**						Y(*6)
		Non-integer						Y(*8)
	Signed	**Integer**						Y(*6)
		Non-integer						Y(*8)
Alphabetic data item				Y(*3) (*4)	Y	Y(*4)		Y(*3)(*4) (*9)
Alphanumeric character item/ **Alphanumeric character function**				Y(*3) (*4)	Y	Y(*4)		Y(*3)(*4) (*9)
External floating-point item								Y(*8)

Receiving side / Sending side			Boolean	String	Byte[]	Char[]	ENUM	Object
Internal floating-point item	**Single precision**							Y(*7)
	Double precision							Y(*8)
National item/ function				Y(*3) (*15)		Y(*15)		Y(*3)(*9) (*15)
Internal Boolean item			Y(*2)					Y(*2) (*10)
External Boolean item			Y(*2)					Y(*2) (*10)
Edited numeric item				Y(*4)				Y(*4)(*9)
Alphanumeric edited item				Y(*4)				Y(*4)(*9)
National edited item				Y(*15)				Y(*9) (*15)
Group item (no type or weakly typed)					Y			
Index data item								
Pointer data item								
Fixed-point literal	**Unsigned**	**Integer**						Y(*6)
		Non-integer						Y(*8)
	Signed	**Integer**						Y(*6)
		Non-integer						Y(*8)
Hexadecimal numeric literal								Y(*6)
Floating-point literal								Y(*8)
Nonnumeric literal **Hexadecimal nonnumeric literal**				Y(*4)	Y	Y(*4)		Y(*4)(*9)
National hexadecimal literal				Y(*15)		Y(*15)		Y(*9) (*15)
Boolean literal			Y(*2)					Y(*2) (*10)
ZERO								
SPACE								
HIGH-VALUE, LOW-VALUE								
QUOTE, symbolic-character								

*1 Substitution is possible if it is a value that can be stored.

*2 Only 1 digit or character can be used.

*3 Trailing spaces of the sending side are removed, that is, they are not sent to the receiving side. When the sending item contains all spaces, NULL is set on the receiving side.

*4 The characters are converted from UTF-8 to UCS-2. If the program is compiled with RCS(ACP) or RCS(ACP-UCS2) the ACP characters are converted to UCS-2.

*5 The substitution is treated as System.Byte.

*6 The substitution is treated as the following types, depending on the number of digits:

```
4 digits or fewer        System.Int16.
5 to 9 digits            System.Int32.
10 or more digits         System.Int64.
```

*7 The substitution is treated as System.Single.

*8 The substitution is treated as System.Double.

*9 "PIC N(1)" items are treated as System.Char, and are substituted. Other items are treated as System.String, and are substituted. If the program is compiled with RCS(ACP), this rule is applied after conversion from ACP to UCS-2.

*10 The substitution is treated as System.Boolean.

*11 The receiving items are treated as signed, and are substituted. For example, when the sending side is Uint32 and the receiving item is Int16 and the value of receiving item is -1 (X"FFFF"), 4294967295(X"FFFFFFFF") is stored to the receiving item.

*12 When the sending item is bigger than the receiving item only the portion of the sending item that fits in the receiving item is transcribed.

*13 The value of the sending item must be within the range of int64 when the digits are considered to represent an integer (i.e. the number you obtain if you remove any decimal point). For example, 999999.9999999000000 produces a compilation error, because 9999999999999000000 is out of the range of int64 (and note, in this case, that the trailing zeroes, that add nothing to the literal with the decimal point, are used by the compiler when constructing the number to check against the int64 range).

*14 When the receiving side is single, it is treated as USAGE COMP-1.
When the receiving side is double, it is treated as USAGE COMP-2.
The rules for the MOVE statement are applied to the substitution.

*15 If the program is compiled with RCS(ACP), ACP is converted to UCS-2.

- Rules for Format 4 (Setting a byte array object as the storage area of the record in the LINKAGE SECTION)

  1. The LINKAGE SECTION of record-1 should be declared as a level 01 or level 77 item.
  2. Identifier-7 should be a byte array object identifier.
  3. Identifier-7 cannot be any of the following:
     - A reference to a field
     - A reference to a property
     - An inline method call
     - An object type conversion

General Rules

- Rule for Format 1

  The SET sentence takes the object reference in identifier-2 and stores it in each identifier-1 in the order in which they are specified.

- Rule for Format 4

  1. The byte array object reference for identifier-7 should be set as the storage area of record-1.
  2. When the byte array object reference for identifier-7 is NULL, NULL should be set as storage area of record-1. In this case, when record-1 is updated/referenced, an exception will occur.

12.8.3.13 STOP

Syntax Rules

The NetCOBOL for .NET compiler does not support the STOP statement specifying literal-1.

12.8.3.14 TRY

This section describes how the TRY statement is used in .NET programming.

Format

TRY imperative-statement-1

[{CATCH[data-name-1] imperative-statement-2} ...]

[FINALLY imperative statement -3]

END-TRY

Syntax Rules

1. Data-name-1 must be an object reference data item that refers to the System.Exception class or its succeeding class.
2. Either the CATCH or FINALLY block must be specified. Both can be specified.
3. The following cannot be specified in imperative-statement-1, imperative-statement-2, or imperative-statement-3:
   a. An out-of-line PERFORM statement.
   b. A SORT statement or MERGE statement with the INPUT PROCEDURE phrase or the OUTPUT PROCEDURE phrase.
   c. The input/output statement if it operates on a file with the LINAGE clause defined in the file section in the method definition.
4. The following statements cannot be specified in imperative-statement-3:
   a. The GO TO statement.
   b. The EXIT PROGRAM or EXIT METHOD statements.
   c. The EXIT TRY statement.
   d. The STOP RUN statement.
   e. The EXIT PERFORM statement.

   However, the EXIT PERFORM statement can be specified if an in-line PERFORM statement corresponding to the EXIT PERFORM statement is specified in imperative-statement-3.

   If a TRY statement is specified in the imperative-statement-3, the EXIT TRY statement can be specified as the imperative-statement-1 or imperative-statement-2 in its TRY statement.
5. If a declarative section exists in a procedure in which the TRY statement is described and the USE statement with INPUT, OUTPUT, I-O, or EXTEND is specified, the input-output statement cannot be specified in imperative-statement-1, imperative-statement-2, or imperative-statement-3.
6. If a declarative section exists in a procedure in which a TRY statement is described and a USE statement of the input/output error procedure is specified, the input-output statement intended for the file specified by the USE statement cannot be specified in imperative-statement-1, imperative-statement-2, or imperative-statement-3.
7. If a declarative section exists in a procedure in which the USE statement of an exception object is specified, the TRY statement cannot be specified.

General Rules

1. The TRY statement blocks the procedure within a specified range. This defines the exception handling and the termination within that range. The range of imperative-statement-1, the range of imperative-statement-2, and the range of imperative-statement-3 are called "TRY block", "CATCH block", and "FINALLY block" respectively.
2. When the TRY block is executed and ends normally, the statements are executed in the following order:
    a. If the FINALLY block is specified, imperative-statement-3 is executed.
    b. Control moves to the end of the END-TRY statement.
3. 3. When the TRY block process ends because an exception is generated, the statements are executed in the following order:
    - If a CATCH statement has been specified for the type of the generated exception, an ancestor type of the generated exception, or a CATCH statement with no data name specified:
        1. The CATCH statement specified for imperative-statement-2 is executed.
        2. If the FINALLY block is specified, imperative-statement-3 is executed.
        3. Control moves to the end of the END-TRY statement.
    - If a CATCH statement has not been specified for the type of the generated exception:
        1. If the FINALLY block is specified, imperative-statement-3 is executed.
        2. The exception is propagated outside of the TRY block.
    - If the TRY block has multiple CATCH statements, the first CATCH block is selected.
4. If an exception is generated and control is shifted to the specific CATCH block, the exception object is assigned to data-name-1 if it is specified.
5. A CATCH block with no data-name-1 specified catches all exceptions that are of types that are subtypes of the System.Exception class. If control shifts to such a CATCH block, the exception object cannot be referred to.
6. If an exception is generated while executing the CATCH block procedure, the CATCH block procedure ends and the statements are executed in the following order:
    1. If the FINALLY block is specified, imperative-statement-3 is executed.
    2. The exception is propagated outside of the TRY statement.
7. If an exception is generated while executing the FINALLY block procedure, the FINALLY block procedure ends and the outside of the TRY statement is notified of an exception.
8. If an embedded SQL statement is specified for imperative-statement-3, no embedded exception declarations are valid for the embedded SQL statement.

12.8.3.15 USE

Format

(Exception object)

USE AFTER EXCEPTION { class-name-1}

Syntax Rules

Class-name-1 should be the name of a class specified in the REPOSITORY paragraph, and which inherits from the System.Exception class.

General Rules

1. The declaratives are searched in the order they are declared in the program. The system executes the first DECLARAVIES section that has a class-name-1, defined in its USE statement, from which the exception object inherits.
2. When the related DECLARATIVE is executed, the predefined object EXCEPTION-OBJECT references the exception object.

12.9 Functions

This section describes the function additions in NetCOBOL for .NET.

Argument Types

When the type of an argument is an object reference, it should be an object reference identifier or an enumeration field reference. The system uses the type of the argument to provide the function value.

Function Types

An object type function is provided for the ENUM functions.

The object reference of the class is an object and category for the object function. The attribute of this function value is provided by the definition of the function. You can write object functions in the following places:

- USING and PROPERTY phrases of the CUSTOM-ATTRIBUTE clause in the SPECIAL-NAMES paragraph.
- Sending side of the SET statement
- Right or left operand of comparisons
- Argument of object functions

12.9.1 ACP-OF Function

The ACP-OF function returns a string of ACP (ANSI Code Page) characters that are equivalent to the characters contained in argument-1.

Format

FUNCTION ACP-OF (argument-1)

Argument

Argument-1 is either an alphanumeric or a national item.

Function Value

1. For the function value, the function converts the characters contained in argument-1 into the corresponding character string represented by ACP (ANSI Code Page). The encoding form of the function value follows that of argument-1.
2. If the character string value contained in argument-1 cannot be represented in ACP, an execution time exception will occur.
3. The length of the function value is the same as the converted character string.

Function Type

The function type is the same as that of argument-1.

12.9.2 DISPLAY-OF Function

The DISPLAY-OF function converts national characters contained in argument-1 to equivalent alphanumeric characters.

Format

FUNCTION DISPLAY-OF (argument-1 [argument-2])

Arguments

1. The class of argument-1 must be national.
2. Argument-2 must be of class alphanumeric and be one character in length.
3. Argument-2 specifies an alphanumeric substitution character for use in conversion of national characters for which there is no corresponding alphanumeric character.

Function Values

1. The function value is a character string that is the result of converting the national characters contained in argument-1 to equivalent alphanumeric characters.
2. If argument-2 is specified, national characters in argument-1 that have no equivalent alphanumeric characters are converted to the substitute character specified in argument-2 in the result.
3. If argument-2 is omitted, and if national characters in argument-1 have no equivalent alphanumeric characters, an execution time exception will occur.
4. The length of the returned value is the number of character positions of usage display required to hold the converted argument and depends on the number of characters contained in argument-1.

Function Value Type

The function value type is alphanumeric.

NetCOBOL-Specific Rule for the DISPLAY-OF Function.

This function cannot be used if the program is compiled with the RCS(ACP) option.

12.9.3 ENUM-OR Function

The ENUM-OR function returns a logical inclusive OR of the list of argument-1's.

Format

FUNCTION ENUM-OR ({ argument-1} ...)

Argument

1. Argument-1 is an enumeration field reference (i.e. field-name OF enum-name) or an object reference of an enumeration.
2. All argument-1's should be of the same enumeration type.

Function Value

The function value is an object reference of the same enumeration type as argument-1.

12.9.4 ENUM-AND Function

The ENUM-AND function returns a logical conjunction of the list of argument-1's.

Format

FUNCTION ENUM-AND ({ argument-1} ...)

Argument

1. Argument-1 is either an enumeration field reference (i.e. field-name OF enum-name) or an object reference of an enumeration.
2. All argument-1's should be the same enumeration type.

Function Value

The function value is an object reference of the same enumeration type as argument-1.

12.9.5 ENUM-NOT Function

The ENUM-NOT function returns a logical negation of the value of argument-1.

Format

FUNCTION ENUM-NOT (argument-1)

Argument

Argument-1 is either an enumeration field reference (i.e. field-name OF enum-name) or an object reference of an enumeration.

Function Value

The function value is an object reference of the same enumeration type as argument-1.

12.9.6 NATIONAL-OF Function

The NATIONAL-OF function converts alphanumeric characters contained in argument-1 to equivalent national characters.

Format

FUNCTION NATIONAL-OF (argument-1 [argument-2])

Arguments

1. The class of argument-1 must be alphanumeric.
2. The class of argument-2 must be national and it must be one character in length.
3. Argument-2 specifies a national substitution character for use in conversion of alphanumeric characters for which there is no corresponding national character.

Function Values

1. The function value is a character string that is the result of converting the alphanumeric characters contained in argument-1 to equivalent national characters.
2. If argument-2 is specified, alphanumeric characters in argument-1 that have no equivalent national characters are converted to the substitute character specified in argument-2 in the result.
3. If argument-2 is omitted, and if alphanumeric characters in argument-1 have no equivalent national characters, an execution time exception will occur.
4. The length of the returned value is the number of character positions of usage national required to hold the converted argument and depends on the number of characters contained in argument-1.

Function Value Type

The function value type is national.

NetCOBOL-Specific Rule for the NATIONAL-OF Function.

This function cannot be used if the program is compiled with the RCS(ACP) option.

12.9.7 UNICODE-OF Function

The UNICODE-OF function returns a UNICODE string equivalent to the characters contained in argument-1.

Format

FUNCTION UNICODE-OF (argument-1)

Argument

Argument-1 is either an alphanumeric or a national item.

Function Value

1. For the function value, the function converts the characters contained in argument-1 into the corresponding character string represented by Unicode. The encoding form of the function value follows that of argument-1.
2. If the character string value contained in argument-1 cannot be represented in UNICODE, an execution time exception will occur.
3. The length of the function value is the same as the converted character string.

Function Type

The function type is the same as that of argument-1.

12.10 Intermediate Results

This section describes the rules of intermediate results specific to NetCOBOL for .NET. For other rules, see Appendix D, "Intermediate Results".

12.10.1 Intermediate Results of Arithmetic Operations

Determining the Temporary Number of Arithmetic Operation Digits

- When the Operand is a BINARY-CHAR/SHORT/LONG/DOUBLE Data Item

  The total number of digits and the number of digits in the decimal part are assumed to be the values shown in the following table. Then the rules shown in the topic titled "When the Operand is a Data Item" in Appendix D, "Intermediate Results" are applied.

Table 12.31 Numbers of Digits for BINARY-CHAR/SHORT/LONG/DOUBLE Data Items

USAGE Clause	Total Number of Digits	Number of Digits in the Decimal Part
BINARY-CHAR UNSIGNED	3	0
BINARY-SHORT SIGNED	5	0
BINARY-LONG SIGNED	10	0
BINARY-DOUBLE SIGNED	19	0

Appendix A List of Reserved Words

The appendix lists the COBOL reserved words.

The asterisks "*" in the column "Remarks" indicate that the specific word has been added for the object-oriented programming function.

o in a cell: Indicates the reserved word is in the language specified for that column.

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
ACCEPT	o	o	o	o	o	o	o	
ACCESS	o	o	o	o	o	o	o	
ACTUAL	o	o	o	o				
ADD	o	o	o	o	o	o	o	
ADDRESS	o	o	o	o				
ADVANCING	o	o	o	o	o	o	o	
AFTER	o	o	o	o	o	o	o	
ALL	o	o	o	o	o	o	o	
ALPHABET	o	o	o	o	o	o		
ALPHABETIC	o	o	o	o	o	o	o	
ALPHABETIC-LOWER	o	o	o	o	o	o		
ALPHABETIC-UPPER	o	o	o	o	o	o		
ALPHANUMERIC	o	o	o	o	o	o		
ALPHANUMERIC-EDITED	o	o	o	o	o	o		
ALSO	o	o	o	o	o	o	o	
ALTER	o	o	o	o		o	o	
ALTERNATE	o	o	o	o	o	o	o	
AND	o	o	o	o	o	o	o	
ANY	o	o	o	o	o	o		
APPLY	o	o	o	o				
ARE	o	o	o	o	o	o	o	
AREA	o	o	o	o	o	o	o	
AREAS	o	o	o	o	o	o	o	
ARITHMETIC	o	o	o	o	o			
AS	o	o	o	o				
ASCENDING	o	o	o	o	o	o	o	
ASSIGN	o	o	o	o	o	o	o	
AT	o	o	o	o	o	o	o	

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
AUTHOR	o	o	o	o		o	o	
AUTO	o	o	o	o				
AUTOMATIC	o	o	o	o				
B-AND	o	o	o	o	o			
B-EXOR	o	o	o	o	o			
B-LESS	o	o	o	o	o			
B-NOT	o	o	o	o	o			
B-OR	o	o	o	o	o			
BASED	o	o	o	o				
BACKGROUND-COLOR	o	o	o	o				
BASED-STORAGE	o	o	o	o				
BEFORE	o	o	o	o	o	o	o	
BEGINNING	o	o	o	o				
BELL	o	o	o	o				
BINARY	o	o	o	o	o	o		
BINARY-CHAR	o	o	o	o				
BINARY-DOUBLE	o	o	o	o				
BINARY-LONG	o	o	o	o				
BINARY-SHORT	o	o	o	o				
BIT	o	o	o	o	o			
BITS	o	o	o	o	o			
BLANK	o	o	o	o	o	o	o	
BLINK	o	o	o	o				
BLOCK	o	o	o	o	o	o	o	
BMP		o	o	o				
BOOLEAN	o	o	o	o	o			
BOTTOM	o	o	o	o	o	o	o	
BY	o	o	o	o	o	o	o	
CALL	o	o	o	o	o	o	o	
CANCEL	o	o	o	o	o	o	o	
CATCH		o	o	o				
CBL	o	o	o	o				
CD	o	o	o	o	o	o	o	

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
CF	o	o	o	o	o	o	o	
CH	o	o	o	o	o	o	o	
CHANGED	o	o	o	o				
CHARACTER	o	o	o	o	o	o	o	
CHARACTERS	o	o	o	o	o	o	o	
CLASS	o	o	o	o	o	o		
CLASS-ID	o	o	o	o				*
CLOCK-UNITS	o	o	o	o		o	o	
CLOSE	o	o	o	o	o	o	o	
COBOL	o	o	o	o		o	o	
CODE	o	o	o	o	o	o	o	
CODE-SET	o	o	o	o	o	o	o	
COLLATING	o	o	o	o	o	o	o	
COLUMN	o	o	o	o	o	o	o	
COMMA	o	o	o	o	o	o	o	
COMMAND	o	o	o	o				
COMMIT	o	o	o	o	o			
COMMON	o	o	o	o	o	o		
COMMUNICATION	o	o	o	o	o	o	o	
COM-REG	o	o	o	o				
COMP	o	o	o	o	o	o	o	
COMP-X	o	o	o	o				
COMP-1	o	o	o	o				
COMP-2	o	o	o	o				
COMP-3	o	o	o	o				
COMP-4	o	o	o	o				
COMP-5	o	o	o	o				
COMP-n					o			
COMPLEX	o	o	o	o				
COMPUTATIONAL	o	o	o	o	o	o	o	
COMPUTATIONAL-X	o	o	o	o				
COMPUTATIONAL-1	o	o	o	o				
COMPUTATIONAL-2	o	o	o	o				

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
COMPUTATIONAL-3	o	o	o	o				
COMPUTATIONAL-4	o	o	o	o				
COMPUTATIONAL-5	o	o	o	o				
COMPUTATIONAL-n					o			
COMPUTE	o	o	o	o	o	o	o	
CONFIGURATION	o	o	o	o	o	o	o	
CONNECT	o	o	o	o	o			
CONSTANT	o	o	o	o				
CONTAINED	o	o	o	o	o			
CONTAINS	o	o	o	o	o	o	o	
CONTENT	o	o	o	o	o	o		
CONTINUE	o	o	o	o	o	o		
CONTROL	o	o	o	o	o	o	o	
CONTROL-CHARACTER	o	o	o	o				
CONTROLS	o	o	o	o	o	o	o	
CONV		o	o					
CONVERSION		o	o					
CONVERTING	o	o	o	o	o	o		
CONV-SIZE		o	o					
CONV-STATUS		o	o					
COPY	o	o	o	o	o	o	o	
CORE-INDEX	o	o	o	o				
CORR	o	o	o	o		o	o	
CORRESPONDING	o	o	o	o		o	o	
COUNT	o	o	o	o	o	o	o	
CRP	o	o	o	o				
CRT	o	o	o	o				
CRT-UNDER	o	o	o	o				
CURRENCY	o	o	o	o	o	o	o	
CURRENT	o	o	o	o	o			
CURSOR	o	o	o	o				
CUSTOM-ATTRIBUTE	o	o	o	o				
DATA	o	o	o	o	o	o	o	

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
DATE	o	o	o	o	o	o	o	
DATE-COMPILED	o	o	o	o		o	o	
DATE-WRITTEN	o	o	o	o		o	o	
DAY	o	o	o	o	o	o	o	
DAY-OF-WEEK	o	o	o	o	o	o		
DB	o	o	o	o	o			
DB-ACCESS-CONTROL-KEY	o	o	o	o	o			
DB-DATA-NAME	o	o	o	o	o			
DB-EXCEPTION	o	o	o	o	o			
DB-RECORD-NAME	o	o	o	o	o			
DB-SET-NAME	o	o	o	o	o			
DB-STATUS	o	o	o	o	o			
DBCS	o	o	o	o				
DE	o	o	o	o	o	o	o	
DEAD-LOCK	o	o	o	o				
DEBUG-CONTENTS	o	o	o	o		o	o	
DEBUG-ITEM	o	o	o	o		o	o	
DEBUG-LINE	o	o	o	o		o	o	
DEBUG-NAME	o	o	o	o		o	o	
DEBUG-SUB-1	o	o	o	o		o	o	
DEBUG-SUB-2	o	o	o	o		o	o	
DEBUG-SUB-3	o	o	o	o		o	o	
DEBUGGING	o	o	o	o	o	o	o	
DECIMAL-POINT	o	o	o	o	o	o	o	
DECLARATIVES	o	o	o	o	o	o	o	
DEFAULT	o	o	o	o	o			
DELEGATE	o	o	o	o				
DELEGATE-ID	o	o	o	o				
DELETE	o	o	o	o	o	o	o	
DELIMITED	o	o	o	o	o	o	o	
DELIMITER	o	o	o	o	o	o	o	
DEPENDING	o	o	o	o	o	o	o	
DESCENDING	o	o	o	o	o	o	o	

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
DESTINATION	o	o	o	o	o	o	o	
DESTINATION-1	o	o	o	o				
DESTINATION-2	o	o	o	o				
DESTINATION-3	o	o	o	o				
DETAIL	o	o	o	o	o	o	o	
DEVICE	o	o	o	o				
DIRECT	o	o	o	o				
DISABLE	o	o	o	o	o	o	o	
DISCONNECT	o	o	o	o	o			
DISJOINING	o	o	o	o				
DISPLAY	o	o	o	o	o	o	o	
DISPLAY-1	o	o	o	o				
DISPLAY-EXIT	o	o	o	o				
DISPLAY-n					o			
DIVIDE	o	o	o	o	o	o	o	
DIVISION	o	o	o	o	o	o	o	
DOWN	o	o	o	o	o	o	o	
DUPLICATE	o	o	o	o	o			
DUPLICATES	o	o	o	o	o	o	o	
DYNAMIC	o	o	o	o	o	o	o	
EDIT-COLOR	o	o	o	o				
EDIT-CURSOR	o	o	o	o				
EDIT-MODE	o	o	o	o				
EDIT-OPTION	o	o	o	o				
EDIT-STATUS	o	o	o	o				
EGCS	o	o	o	o				
EGI	o	o	o	o	o	o	o	
EJECT	o	o	o	o				
ELSE	o	o	o	o	o	o	o	
EMI	o	o	o	o	o	o	o	
EMPTY	o	o	o	o	o			
ENABLE	o	o	o	o	o	o	o	
ENCODING		o	o					

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
END	o	o	o	o	o	o	o	
END-ACCEPT	o	o	o	o				
END-ADD	o	o	o	o	o	o		
END-CALL	o	o	o	o	o	o		
END-COMPUTE	o	o	o	o	o	o		
END-DELETE	o	o	o	o	o	o		
END-DISABLE					o			
END-DISPLAY	o	o	o	o				
END-DIVIDE	o	o	o	o	o	o		
END-ENABLE					o			
END-EVALUATE	o	o	o	o	o	o		
END-EXEC	o	o	o	o				
END-IF	o	o	o	o	o	o		
END-INVOKE	o	o	o	o				*
END-MULTIPLY	o	o	o	o	o	o		
END-OF-PAGE	o	o	o	o	o	o	o	
END-PERFORM	o	o	o	o	o	o		
END-READ	o	o	o	o	o	o		
END-RECEIVE	o	o	o	o	o	o		
END-RETURN	o	o	o	o	o	o		
END-REWRITE	o	o	o	o	o	o		
END-SEARCH	o	o	o	o	o	o		
END-SEND					o			
END-START	o	o	o	o	o	o		
END-STRING	o	o	o	o	o	o		
END-SUBTRACT	o	o	o	o	o	o		
END-TRANSCEIVE					o			
END-TRY		o	o	o				
END-UNSTRING	o	o	o	o	o	o		
END-WRITE	o	o	o	o	o	o		
ENDCOBOL	o	o	o	o				
ENDING	o	o	o	o				
ENTER	o	o	o	o		o	o	

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
ENTRY	o	o	o	o				
ENUM	o	o	o	o				
ENUM-ID	o	o	o	o				
ENVIRONMENT	o	o	o	o	o	o	o	
EOL	o	o	o	o				
EOP	o	o	o	o	o	o	o	
EOS	o	o	o	o				
EQUAL	o	o	o	o	o	o	o	
EQUALS	o	o	o	o	o			
ERASE	o	o	o	o	o			
ERROR	o	o	o	o	o	o	o	
ESI	o	o	o	o	o	o	o	
EVALUATE	o	o	o	o	o	o		
EVERY	o	o	o	o		o	o	
EXACT					o			
EXAMINE	o	o	o	o				
EXCEEDS	o	o	o	o	o			
EXCEPTION	o	o	o	o	o	o	o	
EXCEPTION-OBJECT	o	o	o	o				*
EXCLUSIVE	o	o	o	o	o			
EXEC	o	o	o	o				
EXIT	o	o	o	o	o	o	o	
EXOR	o	o	o	o				
EXTEND	o	o	o	o	o	o	o	
EXTERNAL	o	o	o	o	o	o		
FACTORY	o	o	o	o				*
FALSE	o	o	o	o	o	o		
FD	o	o	o	o	o	o	o	
FETCH	o	o	o	o	o			
FILE	o	o	o	o	o	o	o	
FILE-CONTROL	o	o	o	o	o	o	o	
FILE-LIMIT	o	o	o	o				
FILE-LIMITS	o	o	o	o				

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
FILES	o	o	o	o				
FILLER	o	o	o	o	o	o	o	
FINAL	o	o	o	o	o	o	o	
FINALLY		o	o	o				
FIND	o	o	o	o	o			
FINISH	o	o	o	o	o			
FIRST	o	o	o	o	o	o	o	
FLADD	o	o	o	o				
FLOAT-EXTENDED	o	o	o	o				
FLOAT-LONG	o	o	o	o				
FLOAT-SHORT	o	o	o	o				
FOOTING	o	o	o	o	o	o	o	
FOR	o	o	o	o	o	o	o	
FOREGROUND-COLOR	o	o	o	o				
FORM					o			
FORMAT	o	o	o	o	o			
FORMATTED	o	o	o	o				
FREE	o	o	o	o	o			
FROM	o	o	o	o	o	o	o	
FULL	o	o	o	o				
FUNCTION	o	o	o	o	o			
GENERATE	o	o	o	o	o	o	o	
GET	o	o	o	o	o			
GIVING	o	o	o	o	o	o	o	
GLOBAL	o	o	o	o	o	o		
GO	o	o	o	o	o	o	o	
GOBACK	o	o	o	o				
GREATER	o	o	o	o	o	o	o	
GRID	o	o	o	o				
GROUP	o	o	o	o	o	o	o	
HEADING	o	o	o	o	o	o	o	
HIGHLIGHT	o	o	o	o				
HIGH-VALUE	o	o	o	o	o	o	o	

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
HIGH-VALUES	o	o	o	o	o	o	o	
I-O	o	o	o	o	o	o	o	
I-O-CONTROL	o	o	o	o	o	o	o	
ID	o	o	o	o				
IDENTIFICATION	o	o	o	o	o	o	o	
IF	o	o	o	o	o	o	o	
IN	o	o	o	o	o	o	o	
INCLUDE	o	o	o	o				
INDEX	o	o	o	o	o	o	o	
INDEX-n					o			
INDEXED	o	o	o	o	o	o	o	
INDICATE	o	o	o	o	o	o	o	
INHERITS	o	o	o	o				*
INITIAL	o	o	o	o	o	o	o	
INITIALIZE	o	o	o	o	o	o		
INITIATE	o	o	o	o	o	o	o	
INPUT	o	o	o	o	o	o	o	
INPUT-OUTPUT	o	o	o	o	o	o	o	
INSPECT	o	o	o	o	o	o	o	
INSTALLATION	o	o	o	o		o	o	
INTERFACE	o	o	o	o				
INTERNAL	o	o	o	o				
INTO	o	o	o	o	o	o	o	
INVALID	o	o	o	o	o	o	o	
INVARIANT	o	o	o	o				
INVOKE	o	o	o	o				*
IS	o	o	o	o	o	o	o	
JAPANESE	o	o	o	o				
JOB	o	o	o	o				
JOINING	o	o	o	o				
JUST	o	o	o	o	o	o	o	
JUSTIFIED	o	o	o	o	o	o	o	
KANJI	o	o	o	o				

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
KEEP	o	o	o	o	o			
KEY	o	o	o	o	o	o	o	
LABEL	o	o	o	o		o	o	
LAST	o	o	o	o	o	o	o	
LD	o	o	o	o	o			
LEADING	o	o	o	o	o	o	o	
LEFT	o	o	o	o		o	o	
LEFTLINE	o	o	o	o				
LEFT-JUSTIFY	o	o	o	o				
LENGTH	o	o	o	o	o	o	o	
LESS	o	o	o	o	o	o	o	
LIMIT	o	o	o	o	o	o	o	
LIMITED	o	o	o	o				
LIMITS	o	o	o	o	o	o	o	
LINAGE	o	o	o	o	o	o	o	
LINAGE-COUNTER	o	o	o	o	o	o	o	
LINE	o	o	o	o	o	o	o	
LINE-COUNTER	o	o	o	o	o	o	o	
LINES	o	o	o	o	o	o	o	
LINKAGE	o	o	o	o	o	o	o	
LOCALLY	o	o	o	o	o			
LOCAL-STORAGE		o	o					
LOCK	o	o	o	o	o	o	o	
LOW-VALUE	o	o	o	o	o	o	o	
LOW-VALUES	o	o	o	o	o	o	o	
LOWLIGHT	o	o	o	o				
MANUAL	o	o	o	o				
MEMBER	o	o	o	o	o			
MEMORY	o	o	o	o		o	o	
MERGE	o	o	o	o	o	o	o	
MESSAGE	o	o	o	o	o	o	o	
METHOD	o	o	o	o				*
METHOD-ID	o	o	o	o				*

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
MODE	o	o	o	o	o	o	o	
MODE-1	o	o	o	o				
MODE-2	o	o	o	o				
MODE-3	o	o	o	o				
MODIFY	o	o	o	o	o			
MODULES	o	o	o	o		o	o	
MORE-LABELS	o	o	o	o				
MOVE	o	o	o	o	o	o	o	
MULTICON	o	o	o	o				
MULTICONVERSATION-MODE	o	o	o	o				
MULTIPLE	o	o	o	o	o	o	o	
MULTIPLY	o	o	o	o	o	o	o	
NAMED	o	o	o	o				
NATIONAL	o	o	o	o				
NATIONAL-EDITED	o	o	o	o				
NATIVE	o	o	o	o	o	o	o	
NEGATIVE	o	o	o	o	o	o	o	
NEXT	o	o	o	o	o	o	o	
NO	o	o	o	o	o	o	o	
NOMINAL	o	o	o	o				
NONE	o	o	o	o				
NOT	o	o	o	o	o	o	o	
NOTE	o	o	o	o				
NULL	o	o	o	o	o			
NULLS	o	o	o	o				
NUMBER	o	o	o	o	o	o	o	
NUMERIC	o	o	o	o	o	o	o	
NUMERIC-EDITED	o	o	o	o	o	o		
OBJECT	o	o	o	o				*
OBJECT-COMPUTER	o	o	o	o	o	o	o	
OCCURS	o	o	o	o	o	o	o	
OF	o	o	o	o	o	o	o	
OFF	o	o	o	o		o	o	

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
OMITTED	o	o	o	o		o	o	
ON	o	o	o	o	o	o	o	
ONLY	o	o	o	o	o			
OPEN	o	o	o	o	o	o	o	
OPTIONAL	o	o	o	o	o	o	o	
OR	o	o	o	o	o	o	o	
ORDER	o	o	o	o	o	o		
ORGANIZATION	o	o	o	o	o	o	o	
OTHER	o	o	o	o	o	o		
OTHERWISE	o	o	o	o				
OUTPUT	o	o	o	o	o	o	o	
OVERFLOW	o	o	o	o	o	o	o	
OVERLINE	o	o	o	o				
OVERRIDE	o	o	o	o				*
OWNER	o	o	o	o	o			
PACKED-DECIMAL	o	o	o	o	o	o		
PADDING	o	o	o	o	o	o		
PAGE	o	o	o	o	o	o	o	
PAGE-COUNTER	o	o	o	o	o	o	o	
PARAGRAPH					o			
PARTIAL		o	o	o				
PASSWORD	o	o	o	o				
PERFORM	o	o	o	o	o	o	o	
PF	o	o	o	o	o	o	o	
PH	o	o	o	o	o	o	o	
PIC	o	o	o	o	o	o	o	
PICTURE	o	o	o	o	o	o	o	
PLUS	o	o	o	o	o	o	o	
POINTER	o	o	o	o	o	o	o	
POSITION	o	o	o	o	o	o	o	
POSITIONING	o	o	o	o				
POSITIVE	o	o	o	o	o	o	o	
PREFIX	o	o	o	o				

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
PRESENT	o	o	o	o	o			
PREVIOUS	o	o	o	o				
PRINTING	o	o	o	o	o	o	o	
PRIOR	o	o	o	o	o			
PRIVATE	o	o	o	o				
PROCEDURE	o	o	o	o	o	o	o	
PROCEDURES	o	o	o	o		o	o	
PROCEED	o	o	o	o		o	o	
PROCESSING	o	o	o	o				
PROGRAM	o	o	o	o	o	o	o	
PROGRAM-ID	o	o	o	o	o	o	o	
PROGRAM-STATUS	o	o	o	o				
PROMPT	o	o	o	o				
PROPERTY	o	o	o	o				*
PROTECTED	o	o	o	o	o			
PROTOPTYPE	o	o	o	o				*
PUBLIC	o	o	o	o				
PURGE	o	o	o	o	o	o		
QUEUE	o	o	o	o	o	o	o	
QUOTE	o	o	o	o	o	o	o	
QUOTES	o	o	o	o	o	o	o	
RAISE	o	o	o	o				*
RAISING	o	o	o	o				*
RANDOM	o	o	o	o	o	o	o	
RANGE	o	o	o	o				
RD	o	o	o	o	o	o	o	
READ	o	o	o	o	o	o	o	
READY	o	o	o	o	o			
REALM	o	o	o	o	o			
RECEIVE	o	o	o	o	o	o	o	
RECONNECT	o	o	o	o	o			
RECORD	o	o	o	o	o	o	o	
RECORD-NAME	o	o	o	o	o			

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
RECORD-OVERFLOW	o	o	o	o				
RECORDING	o	o	o	o				
RECORDS	o	o	o	o	o	o	o	
RECURSIVE		o	o					
REDEFINES	o	o	o	o	o	o	o	
REEL	o	o	o	o	o	o	o	
REFERENCE	o	o	o	o	o	o		
REFERENCES	o	o	o	o		o	o	
RELATION	o	o	o	o	o			
RELATIVE	o	o	o	o	o	o	o	
RELEASE	o	o	o	o	o	o	o	
RELOAD	o	o	o	o				
REMAINDER	o	o	o	o	o	o	o	
REMARKS	o	o	o	o				
REMOVAL	o	o	o	o	o	o	o	
RENAMES	o	o	o	o	o	o	o	
REORG-CRITERIA	o	o	o	o				
REPEATED	o	o	o	o	o			
REPLACE	o	o	o	o	o	o		
REPLACING	o	o	o	o	o	o	o	
REPORT	o	o	o	o	o	o	o	
REPORTING	o	o	o	o	o	o	o	
REPORTS	o	o	o	o	o	o	o	
REPOSITORY	o	o	o	o				*
RERUN	o	o	o	o		o	o	
REQUIRED	o	o	o	o				
RESERVE	o	o	o	o	o	o	o	
RESET	o	o	o	o	o	o	o	
RESUME		o	o	o				
RETAINING	o	o	o	o	o			
RETRIEVAL	o	o	o	o	o			
RETURN	o	o	o	o	o	o	o	
RETURNING	o	o	o	o				

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
RETURN-CODE	o	o	o	o				
REVERSE-VIDEO	o	o	o	o				
REVERSED	o	o	o	o		o	o	
REWIND	o	o	o	o	o	o	o	
REWRITE	o	o	o	o	o	o	o	
RF	o	o	o	o	o	o	o	
RH	o	o	o	o	o	o	o	
RIGHT	o	o	o	o	o	o	o	
RIGHT-JUSTIFY	o	o	o	o				
ROLL-OUT	o	o	o	o				
ROLLBACK	o	o	o	o	o			
ROUNDED	o	o	o	o	o	o	o	
RUN	o	o	o	o	o	o	o	
SA	o	o	o	o				
SAME	o	o	o	o	o	o	o	
SAVED-AREA	o	o	o	o				
SCREEN	o	o	o	o				
SD	o	o	o	o	o	o	o	
SEARCH	o	o	o	o	o	o	o	
SECTION	o	o	o	o	o	o	o	
SECURITY	o	o	o	o		o	o	
SECURE	o	o	o	o				
SEEK	o	o	o	o				
SELF	o	o	o	o				*
SEGMENT	o	o	o	o	o	o	o	
SEGMENT-LIMIT	o	o	o	o		o	o	
SELECT	o	o	o	o	o	o	o	
SELECTED	o	o	o	o				
SELECTIVE	o	o	o	o				
SEND	o	o	o	o	o	o	o	
SENTENCE	o	o	o	o	o	o	o	
SEPARATE	o	o	o	o	o	o	o	
SEQUENCE	o	o	o	o	o	o	o	

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
SEQUENTIAL	o	o	o	o	o	o	o	
SERVICE	o	o	o	o				
SESSION	o	o	o	o				
SESSION-ID					o			
SET	o	o	o	o	o	o	o	
SHARED	o	o	o	o	o			
SHIFT-IN	o	o	o	o				
SHIFT-OUT	o	o	o	o				
SIGN	o	o	o	o	o	o	o	
SIMPLE	o	o	o	o				
SINGLE	o	o	o	o				
SIZE	o	o	o	o	o	o	o	
SKIP1	o	o	o	o				
SKIP2	o	o	o	o				
SKIP3	o	o	o	o				
SORT	o	o	o	o	o	o	o	
SORT-CONTROL	o	o	o	o				
SORT-CORE-SIZE	o	o	o	o				
SORT-FILE-SIZE	o	o	o	o				
SORT-MESSAGE	o	o	o	o				
SORT-MERGE	o	o	o	o	o	o	o	
SORT-MODE-SIZE	o	o	o	o				
SORT-RETURN	o	o	o	o				
SORT-STATUS	o	o	o	o				
SOURCE	o	o	o	o	o	o	o	
SOURCE-COMPUTER	o	o	o	o	o	o	o	
SPACE	o	o	o	o	o	o	o	
SPACE-FILL	o	o	o	o				
SPACES	o	o	o	o	o	o	o	
SPECIAL-NAMES	o	o	o	o	o	o	o	
STANDARD	o	o	o	o	o	o	o	
STANDARD-1	o	o	o	o	o	o	o	
STANDARD-2	o	o	o	o	o	o		

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
START	o	o	o	o	o	o	o	
STATIC	o	o	o	o				
STATION	o	o	o	o				
STATIONS	o	o	o	o				
STATUS	o	o	o	o	o	o	o	
STOP	o	o	o	o	o	o	o	
STORE	o	o	o	o	o			
STRING	o	o	o	o	o	o	o	
SUB-QUEUE-1	o	o	o	o	o	o	o	
SUB-QUEUE-2	o	o	o	o	o	o	o	
SUB-QUEUE-3	o	o	o	o	o	o	o	
SUBRANGE	o	o	o	o				
SUB-SCHEMA	o	o	o	o	o			
SUBSCHEMA-NAME	o	o	o	o				
SUBTRACT	o	o	o	o	o	o	o	
SUCCESSIVE	o	o	o	o				
SUFFIX	o	o	o	o				
SUM	o	o	o	o	o	o	o	
SUPER	o	o	o	o				*
SUPPRESS	o	o	o	o	o	o	o	
SYMBOLIC	o	o	o	o	o	o	o	
SYNC	o	o	o	o		o	o	
SYNCHRONIZED	o	o	o	o		o	o	
TABLE	o	o	o	o	o	o	o	
TALLY	o	o	o	o				
TALLYING	o	o	o	o	o	o	o	
TAPE	o	o	o	o	o	o	o	
TENANT	o	o	o	o				
TENNANT					o			
TERMINAL	o	o	o	o	o	o	o	
TERMINATE	o	o	o	o	o	o	o	
TEST	o	o	o	o	o	o		
TEXT	o	o	o	o	o	o	o	

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
THAN	o	o	o	o	o	o	o	
THEN	o	o	o	o	o	o		
THROUGH	o	o	o	o	o	o	o	
THRU	o	o	o	o	o	o	o	
TIME	o	o	o	o	o	o	o	
TIMEOUT					o			
TIMES	o	o	o	o	o	o	o	
TITLE	o	o	o	o				
TO	o	o	o	o	o	o	o	
TOP	o	o	o	o	o	o	o	
TRACE	o	o	o	o				
TRACK	o	o	o	o				
TRACK-AREA	o	o	o	o				
TRACK-LIMIT	o	o	o	o				
TRACK-OVERFLOW	o	o	o	o				
TRACKS	o	o	o	o				
TRAILING	o	o	o	o	o	o	o	
TRAILING-SIGN	o	o	o	o				
TRANSACTION	o	o	o	o				
TRANSCEIVE					o			
TRUE	o	o	o	o	o	o		
TRY		o	o	o				
TYPE	o	o	o	o	o	o	o	
TYPEDEF	o	o	o	o				
UNDERLINE	o	o	o	o				
UNEQUAL	o	o	o	o	o			
UNICODE1		o	o	o				
UNIT	o	o	o	o	o	o	o	
UNIVERSAL	o	o	o	o				*
UNLOCK	o	o	o	o				
UNSTRING	o	o	o	o	o	o	o	
UNTIL	o	o	o	o	o	o	o	
UP	o	o	o	o	o	o	o	

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
UPDATE	o	o	o	o	o			
UPON	o	o	o	o	o	o	o	
USAGE	o	o	o	o	o	o	o	
USAGE-MODE	o	o	o	o	o			
USE	o	o	o	o	o	o	o	
USING	o	o	o	o	o	o	o	
VALID	o	o	o	o	o			
VALIDATE	o	o	o	o	o			
VALUE	o	o	o	o	o	o	o	
VALUES	o	o	o	o	o	o	o	
VARYING	o	o	o	o	o	o	o	
WAIT	o	o	o	o	o			
WHEN	o	o	o	o	o	o	o	
WHEN-COMPILED	o	o	o	o				
WITH	o	o	o	o	o	o	o	
WITHIN	o	o	o	o	o			
WORDS	o	o	o	o		o	o	
WORKING-STORAGE	o	o	o	o	o	o	o	
WRITE	o	o	o	o	o	o	o	
WRITE-ONLY	o	o	o	o				
ZERO	o	o	o	o	o	o	o	
ZERO-FILL	o	o	o	o				
ZEROES	o	o	o	o	o	o	o	
ZEROS	o	o	o	o	o	o	o	
+	o	o	o	o		o	o	
-	o	o	o	o		o	o	
*	o	o	o	o		o	o	
/	o	o	o	o		o	o	
**	o	o	o	o		o	o	
>	o	o	o	o		o	o	
<	o	o	o	o		o	o	
=	o	o	o	o		o	o	
>=	o	o	o	o		o		

Word	NetCOBOL				CODASYL 1988 (reference)	ANSI 1985 (or JIS 1992)	ANSI 1974	Remarks
	[Linux]	[Win32] [Winx64] [Linux64]	[.NET]	Others				
<=	o	o	o	o		o		
&	o	o	o	o				
->	o	o	o	o				
*>	o	o	o	o				
::	o	o	o	o				*

F

Appendix B System Quantitative Restrictions

This appendix provides the quantitative restrictions in the COBOL processing system. The restrictions as described here are logical values, and restrictions on program operation imposed by the operating system and the I/O control system used.

B.1 Reference Format

Item	Value
Maximum length of variable-format (in bytes)	251
Maximum length of free-formats (in bytes)	251

B.2 Data Division

Item	Value	
	[Win16]	Other
Maximum length of data item (in bytes)	64770	2147483647
Maximum length of alphabetic and alphanumeric data item (in characters)	64770	2147483647
Maximum length of alphanumeric edited data item (in characters)	64770	2147483647
Maximum length of Boolean data item (in characters)	64770	2147483647
Maximum length of numeric edited data item (in characters)	160	
Maximum length of national data item (in characters)	32385	1073741823 (*1)
Maximum length of national edited data item (in characters)	32385	1073741823 (*1)
Maximum value of a subscript	64770	2147483647
Maximum value of dimension of subscript	7	
Maximum repeat count of OCCURS clause	2147483647	
Maximum length of a key item specified in the KEY IS phrase of OCCURS clause (in bytes)	256	
Maximum number of keys for a table element	2147483647	
Maximum number of index-names with the INDEX BY phrase in OCCURS clause	60	
Maximum depth of implicit pointer qualification	7	

*1: When the encoding form is UTF-32, the value is 536870911.

B.3 Procedure Division

Item	Value
Maximum size of identifier-1 when the function name CONSOLE is assigned to mnemonic-name in "ACCEPT identifier-1 FROM mnemonic-name" (in bytes)	No restriction
Maximum number of identifiers in an INITIALIZE statement	No restriction
Number of data items or variable-length items subscripted, pointed, or reference modified using the identifier in an INITIALIZE procedure	No restriction
Maximum size of identifier-6 and identifier-7 in the "INSPECT CONVERTING identifier-6 TO identifier-7" statement (in bytes) a. When identifier-6 and identifier-7 are neither national data item nor national edited data item	a. 256 b. No restriction

Item	Value
b. When identifier-6 and identifier-7 are national data item and/or national edited data item	
Maximum value of identifier-1 or integer-1 in the "PERFORM identifier-1/integer-1 TIMES" statement	No restriction
The maximum size (bytes) of identifier-1 when the function name ENVIRONMENT-NAME is associated with the mnemonic name of the "DISPLAY identifer-1 UPON mnemonic" statement.	512
The maximum size (bytes) of identifier-1 when the function name ENVIRONMENT-VALUE is associated with the mnemonic name of the "DISPLAY identifer-1 UPON mnemonic" statement.	2047
The maximum size (bytes) of identifier-1 when the function name ENVIRONMENT-VALUE is associated with the mnemonic name of the "ACCEPT identifer-1 UPON mnemonic" statement.	2047

B.4 Sequential File

Item	Value
Maximum record length (in bytes)	32760
Maximum number of lines per logical page specified in LINAGE clause	No restriction

B.5 Relative File

Item	Value
Maximum record length (in bytes)	32760

B.6 Indexed File

Item	Value
Maximum record length (in bytes)	32760
Maximum number of data-names in RECORD KEY clause	254 (*1)
Maximum value of the sum of length of data-name in RECORD KEY clause (in bytes)	254 (*2)
Maximum number of data-name in ALTERNATE RECORD KEY clause	254 (*1)
Maximum value of the sum of length of data-name in an ALTERNATE RECORD KEY clause (in bytes)	254 (*2)
Maximum number of ALTERNATE RECORD KEY clauses	125

*1: The total number of data-name in the RECORD KEY and ALTERNATE RECORD KEY clauses are maximum 255.

*2: The total length of data-name in the RECORD KEY and ALTERNATE RECORD KEY clauses are maximum 255.

B.7 Inter-program Communication

Item	Value
Maximum number of header parameters in the PROCEDURE DIVISION	No restriction
Maximum number of parameters having the USING phrase in the CALL statement	No restriction

Item	Value
Maximum number of parameters having the USING phrase in the ENTRY statement	No restriction
Length of characters on command line	260 (argument only 255)

B.8 Sort and Merge

Item	Value	
	[Win16][Win32]	Other
Maximum record length (in bytes)	21484	32760
Maximum number of key data items specified in the SORT statement and the MERGE statement	64	
Maximum value of the sum of length of key data items specified in the SORT statement and the MERGE statement (in bytes)	Maximum record length	
Maximum length of alphabetic and alphanumeric data item specified as a key data item in the SORT statement and the MERGE statement (in characters)	16382	
Maximum length of national data item specified as a key data item in the SORT statement and the MERGE statement (in characters)	8191	
Maximum number of input files specified with USING in the SORT statement and the MERGE statement	16	

B.9 Source Statement Manipulation

Item	Value
Maximum length of text words in the COPY statement (in characters)	324
Maximum length of word-4 in "COPY ... JOINING word-4" (in characters)	28
Maximum length of text word in the REPLACE statement	324

B.10 Presentation File

Item	Value
Maximum record length (in bytes)	32767

B.11 Object-Oriented Programming

Item	Value
Maximum number of parameters in the PROCEDURE DIVISION header	No restriction
Maximum number of parameters in the USING phrase of the INVOKE statement	No restriction

Appendix C Code Tables

This appendix lists internal codes for characters in the following code systems:

EBCDIC (Extended Binary Coded Decimal Interchange Code)

ASCII (American National Standard Code for Information Interchange X3.4 - 1968)

JIS 8-bit Code (conforming to ISO 640)

EBCDIC, ASCII, and JIS 8-bit codes correspond to EBCDIC, STANDARD-1, and STANDARD-2, respectively in the ALPHABET clause in the SPECIAL-NAMES paragraph. Read the tables as follows:

A character is represented using a one-byte internal code. Characters 0 to F in the "high-order 4 bits" and "low-order 4 bits" columns in the table represent bits 1 to 4 and bits 5 to 8 in hexadecimal notation. Characters in the table are represented in combinations of the high-order 4 bits and low-order 4 bits.

The location of characters in the table shows the collating sequence. Characters in the left columns are greater than those in the right columns; and characters in lower rows are greater than those in higher rows.

C.1 Internal Codes for EBCDIC Characters

Low-order 4 Bits	High-order 4 Bits															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					SP	&	-			(*1)			{	}	$	0
1						(*1)			a (*3)	j (*3)	~		A	J		1
2						(*1)			b (*3)	k (*3)	s (*3)		B	K	S	2
3						(*1)			c (*3)	l (*3)	t (*3)		C	L	T	3
4						(*1)			d (*3)	m (*3)	u (*3)		D	M	U	4
5						(*1)			e (*3)	n (*3)	v (*3)		E	N	V	5
6					(*1)	(*1)			f (*3)	o (*3)	w (*3)		F	O	W	6
7					(*1)				g (*3)	p (*3)	x (*3)		G	P	X	7
8					(*1)				h (*3)	q (*3)	y (*3)		H	Q	Y	8
9					(*1)			`	i (*3)	r (*3)	z (*3)		I	R	Z	9
A					(*2)	!	¦	:	(*1)	(*1)	(*1)	(*1)				
B					.	$ \	,	#				(*1)				
C					<	*	%	@	(*1)		(*1)	(*1)				
D					(	)	_	`	(*1)	(*1)	(*1)	(*1)				
E					+	;	>	=	(*1)	(*1)	(*1)	(*1)				
F					¦	Ø	?	"	(*1)	(*1)	(*1)	(*1)				

*1: Japanese character

*2: The pound sign

*3: Two kinds of characters are represented with the same internal code, one for lowercase, the other for Japanese Kana.

C.2 Internal Codes for ASCII Characters

Low-order 4 Bits	High-order 4 Bits							
	0	1	2	3	4	5	6	7
0			SP	0	@	P	`	p
1			!	1	A	Q	a	q
2			"	2	B	R	b	r
3			#	3	C	S	c	s
4			$	4	D	T	d	t
5			%	5	E	U	e	u
6			&	6	F	V	f	v
7			`	7	G	W	g	w
8			(	8	H	X	h	x
9			)	9	I	Y	I	y
A			*	:	J	Z	j	z
B			+	;	K	[	k	{
C			-	<	L	\	l	|
D			-	=	M	]	m	}
E			.	>	N	^	n	~
F			/	?	O	_	o	

C.3 Internal Codes for JIS 8-Bit Code Characters

Low-order 4 Bits	High-order 4 Bits															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			SP	0	@	P	`	p				(*1)	(*1)	(*1)		
1			!	1	A	Q	a	q			(*1)	(*1)	(*1)	(*1)		
2			"	2	B	R	b	r			(*1)	(*1)	(*1)	(*1)		
3			#	3	C	S	c	s			(*1)	(*1)	(*1)	(*1)		
4			$	4	D	T	d	t			(*1)	(*1)	(*1)	(*1)		
5			%	5	E	U	e	u			(*1)	(*1)	(*1)	(*1)		
6			&	6	F	V	f	v			(*1)	(*1)	(*1)	(*1)		
7			`	7	G	W	g	w			(*1)	(*1)	(*1)	(*1)		
8			(	8	H	X	h	x			(*1)	(*1)	(*1)	(*1)		
9			)	9	I	Y	I	y			(*1)	(*1)	(*1)	(*1)		
A			*	:	J	Z	j	z			(*1)	(*1)	(*1)	(*1)		
B			+	;	K	[	k	{			(*1)	(*1)	(*1)	(*1)		
C			,	<	L	\	l	|			(*1)	(*1)	(*1)	(*1)		
D			-	=	M	]	m	}			(*1)	(*1)	(*1)	(*1)		

Low-order 4 Bits	High-order 4 Bits															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E			.	>	N	^	n	~			(*1)	(*1)	(*1)	(*1)		
F			/	?	O	_	o				(*1)	(*1)	(*1)	(*1)		

*1: Japanese character

F

Appendix D Intermediate Results

A temporary operational result is produced while the value of an arithmetic statement or arithmetic expression or a function value is determined. It is called an "intermediate result". The attribute and precision of the intermediate result are determined according to the rules described in this appendix. These rules also apply to intermediate results of a data item, a function value, an intermediate result, and two intermediate results.

If the intermediate result value exceeds its precision at execution time, the intermediate result will be truncated. Therefore, if a longer data item is specified in the arithmetic statement or arithmetic expression operand or in the function argument, unexpected truncation may occur during evaluation of the results.

D.1 Attribute and Precision of the Intermediate Result

There are two intermediate result attributes: fixed floating-point attribute and floating-point attribute.

An intermediate result having the fixed-point attribute is handled as binary, packed decimal, or zoned decimal data item. The precision of the fixed-point attribute is represented using the total number of digits and the number of digits in the decimal-part.

The intermediate result having the floating-point attribute is handled as an internal floating-point data item. Precision of a floating-point attribute is either long precision or short precision. An intermediate result having long precision is handled as a long precision floating-point data item. An intermediate result having short precision is handled as a short precision floating-point data item.

The attribute and precision of intermediate results depend on the operation mode, the arithmetic operation and function types.

D.2 Intermediate Results of Arithmetic Operations

Intermediate results of arithmetic operations (addition, subtraction, multiplication, and division) have the following attributes:

In 18-digit compatible operation mode

- When both operands in an arithmetic operation are numeric data items, numeric literals, or a fixed point intermediate result from a prior operation, the intermediate result will be fixed point.
- When at least one operand of an arithmetic operation is a floating-point data item, floating-point literal, or a floating-point intermediate result from a prior operation, the intermediate result will be floating-point.

In 31-digit extension operation mode

- If an arithmetic statement or an arithmetic expression includes a floating-point element, the intermediate result of all arithmetic operations in the arithmetic statement or the arithmetic expression will be floating-point.
- If the relation condition includes a floating-point element, the intermediate result of all arithmetic operations in the arithmetic statement or the arithmetic expression will be floating-point.

  For the EVALUATE statement, the intermediate result is determined by the combination of selection subjects and selection objects.
  In the example shown below, A, B, C and E are fixed-point and D is floating point.

```
EVALUATE  A + B
WHEN   C * D + 1
   :
WHEN   E
   :
END-EVALUATE
```

  - Comparison between "A + B" and "C * D + 1":

    Since D is floating point, the intermediate results of "A + B" and "C * D + 1" are floating point.

- Comparison between "A + B" and "E":

  Since A, B and E are fixed-point, the intermediate result of "A + B" is fixed-point.

Note

- 31-digit extension operation mode is available in [Winx64] and [Linux64].
- When an arithmetic expression includes a function, the argument of the function is considered in determining the attribute of the intermediate result.
- A floating-point element includes the following:
  - A floating-point data item
  - A floating-point literal
  - Exponentiation where the exponent is not an integer
  - Functions where the attribute of the intermediate result is always floating-point

D.2.1 Precision of the Intermediate Results of Arithmetic Operations Having the Fixed Point Attribute

Precision of an intermediate result of an arithmetic operation having the fixed point attribute is represented by the total number of digits and the number of digits in the decimal part. The total number of digits of the intermediate result does not include picture character string P.

Precision is determined using the procedures below.

1. The reference number of digits in the decimal part is determined from the operand attributes in a statement containing any arithmetic operation or expression.
2. The temporary number of digits of an arithmetic operation is determined from the operands and number of digits in the arithmetic operations decimal part.
3. The precision of the intermediate result of an arithmetic operation is determined by the temporary number of digits in an arithmetic operation and the reference number of digits in the decimal-part.

Determining the Reference Number of Digits in the Decimal-part

First, the reference number of digits in the decimal part is determined. The "reference number of digits in the decimal-part" is the maximum number of digits in the decimal part that will assure adequate storage for an intermediate result.

For an Arithmetic Statement or Arithmetic Expression

All operands in arithmetic statements and arithmetic expressions (including destination items) are used. Use the maximum number of digits in the decimal part operand as the reference number of digits in the decimal part. However, when using arithmetic statements and expressions, floating-point data items and any operands used as an exponent in an exponent or division divisor and as function arguments should be excluded. When ROUNDED is specified in the destination, increase the number of digits in the decimal part of the destination item and use that result for the destination item. If no operand is used to determine the reference number of digits in the decimal part, 0 is used as the reference number of digits in the decimal part.

For Relation Conditions or the EVALUATE Statement

Use any operand in an optional body or selectable pair in a relation condition or EVALUATE statement. Of these operands, the maximum number of digits in the decimal-part is used as the reference number of digits in the decimal-part. However, in the relation condition or EVALUATE statement, any operands used as an exponent in the square or division divisor and as function arguments should be excluded. If no operand is used to determine the reference number of digits in the decimal-part, 0 is used as the reference number of digits in the decimal-part.

Determining the Temporary Number of Arithmetic Operation Digits

After determining the reference number of digits in the decimal part, the temporary number of arithmetic operation digits is determined. The "temporary number of arithmetic operation digits" is the maximum number of digits of the arithmetic operation; this is used to obtain the digits of the finalized intermediate result. This number is used to obtain the number of final digits of the intermediate result.

The temporary number of arithmetic operation digits is determined using the actual values shown below(*).

* : When the sum of the number of digits of the arithmetic operands is 30 digits or more, the actual values shown below are not used, and the number of digits is determined according to the number of defined digits and the number of digits in the decimal part.

When the Operand is a Data Item

Except for the divisor, use the maximum value that the string in the PICTURE clause for data item can hold. For example, for "PIC S9(5)V99", use 99999.99.

For the divisor, use the minimum value out of the positive numbers that the string in the PICTURE clause for data item can hold. For example, for "PIC S9(5)V99," use 0.01.

When the Operand is a BINARY-CHAR/SHORT/LONG/DOUBLE data item, the total number of digits and the number of digits in the decimal part are assumed to be the values shown in the following table.

USAGE Clause	Total Number of Digits	Number of Digits in the Decimal Part
BINARY-CHAR UNSIGNED	3	0
BINARY-SHORT SIGNED	5	0
BINARY-LONG SIGNED	10	0
BINARY-DOUBLE SIGNED	19	0

When the Operand is a Literal

The absolute value of the literal value is used.

When the Operand is an Intermediate Result

Operational results are determined together with the precision of the intermediate result. This result is used in the following:

- When the temporary number of arithmetic operation digits is 30 or less, and the temporary number is calculated.
- When the temporary number of arithmetic operation digits is over 29, use the maximum value (30 digits 99 .... 99) having the precision of the intermediate result.

Number of Digits in the Decimal-part in the Temporary Number of Arithmetic Operation Digits

The table below lists the number of digits in the decimal-part in the temporary number of arithmetic operation digits.

Arithmetic Operation	Number of Digits in Decimal-part
Addition (OP1 + OP2) Subtraction (OP1 - OP2)	MAX(OP1d, OP2d)
Multiplication (OP1 * OP2)	OP1d+OP2d
Division (OP1/OP2)	MAX(Bd, OP1d-OP2d)

Note

OP1d and OP2d indicate the number of digits in the decimal-part of OP1 and OP2, respectively.

Bd indicates the reference number of digits in the decimal-part.

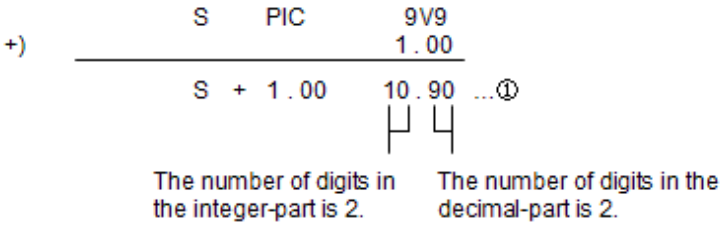
This table can be used when the temporary number of arithmetic operation digits is determined according to the number of defined digits and the number of digits in the decimal part.

Example of Determining the Temporary Number of Arithmetic Operation Digits (Example 1)

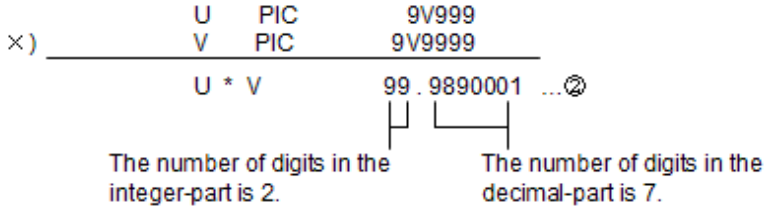
```
S PIC S9V9.
T PIC S9V99.
U PIC S9V999.
V PIC S9V9999.
X PIC S9999V9999.

COMPUTE X = S + 1.00 + U * V.
```

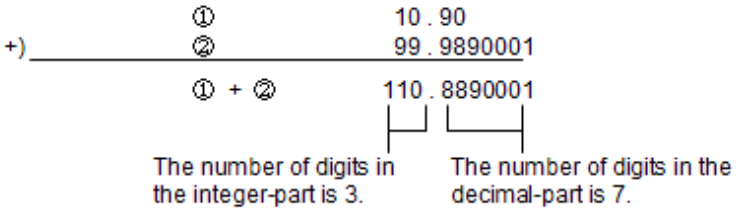
a. Precision of the intermediate result of S + 1.00 is determined as detailed below:



b. Precision of the intermediate result of U * V is determined as shown below:



c. Precision of the intermediate result of S + 1.00 plus intermediate result of U * V is determined as detailed below:

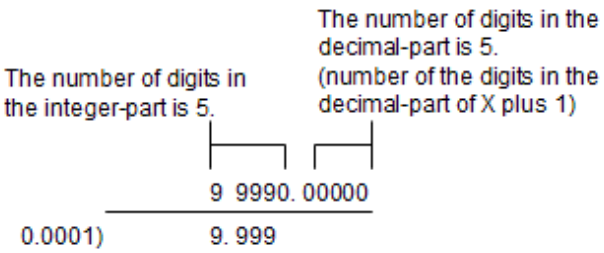


Example of Determining the Temporary Number of Arithmetic Operation Digits (Example 2)

```
DIVIDE  U  BY  V  GIVING  X  ROUNDED .

Assume the same operand attributes as in example 1.
```

Precision of the intermediate result of the arithmetic operation U/V is determined as shown below:

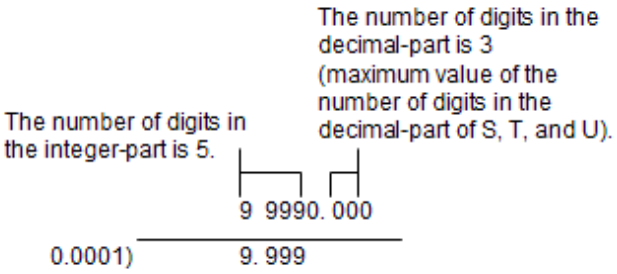


Example of Determining the Temporary Number of Arithmetic Operation Digits (Example 3)

```
COMPUTE  S = S + T + ( S * U ) / ( U / V )

Assume the same operand attributes as in example 1.
```

Precision of the intermediate result of the arithmetic operation U/V is determined as shown below:



Determining Precision of the Intermediate Result of Arithmetic Operations

The precision of the arithmetic operation is determined after the temporary number of arithmetic operation digits is determined.. First, it must be determined if the temporary number of arithmetic operation digits exceeds the *maximum digit* (*):

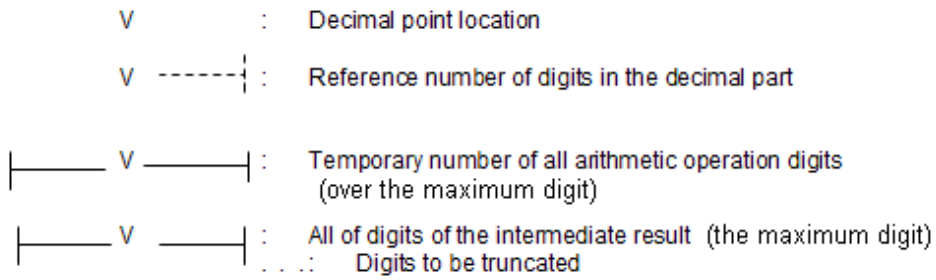
- When the temporary number of arithmetic operation digits is equal to or less than the *maximum digit* , the precision of the intermediate result is used.
- When the temporary number of arithmetic operation digits exceeds the *maximum digit*, truncate the number of digits to the *maximum digit* to obtain the precision of the intermediate result.

  * The *maximum digit* depends on the operation mode. Note that 31-digit extension operation mode is specific to [Winx64] and [Linux64].

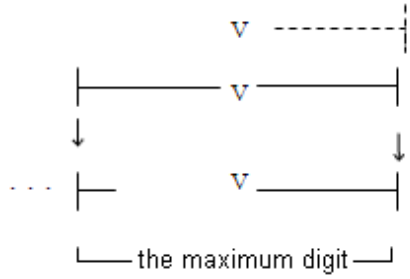
  - 18-digit compatible operation mode : the *maximum digit* is 30
  - 31-digit extension operation mode : the *maximum digit* is 38

Rules for truncation when the temporary number of arithmetic operation digits exceeds *the maximum digit* is the following:

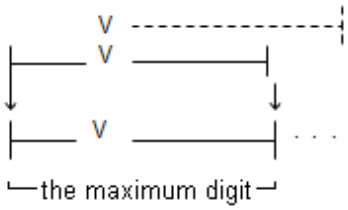
Symbols used throughout the description mean the following:



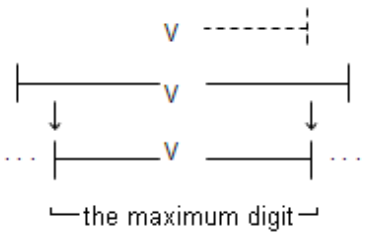
a. When the number of digits in the decimal-part is less than the reference number of digits in the decimal-part, then the number of digits in the decimal-part determined above is used, and the higher digits of the integer part are truncated.



b. When the number of digits in the decimal-part is greater than the reference number of digits in the decimal-part, and the sum of the number of digits in the integer-part and the reference number of digits in the decimal-part is *the maximum digit* or less, then the number of digits in the integer-part determined above is used, and the lower digits of the decimal-part are truncated.



c. When the number of digits in the decimal-part is greater than the reference number of digits in the decimal-part, and the sum of the number of digits in the integer-part and the reference number of digits in the decimal-part exceeds *the maximum digit*, then the reference number of digits in the decimal-part is used as the number of digits in the decimal-part, and the lower digits in the decimal-part and higher digits in the integer-part are truncated.



D.2.2 Intermediate Result in Arithmetic Operations Having the Floating-point Attribute

Precision of the intermediate result of arithmetic operations having the floating-point attribute is determined by the following:

- When at least one operand of an arithmetic operation is long precision, the intermediate result will have long precision.
- When both operands in an arithmetic operation are short precision, the intermediate result will have short precision.

D.3 Intermediate Result of the Exponent

The intermediate result of the exponent has the following attributes:

- When the exponent is an integer, the intermediate result of the exponent has the fixed point attribute.
- When the exponent is not an integer, the intermediate result of the exponent has the floating-point attribute.

D.3.1 Intermediate Result of the Exponent Having Fixed-point Attribute

Precision of the intermediate result of the exponent having the fixed point attribute depends on whether the exponent is a literal or variable.

When the Exponent is a Literal

When the exponent is a literal (integer), the exponent (A ** B) is expanded to the formulas shown below. At this point, the intermediate result of each multiplication (T1, T2, ..., Tn) and the intermediate result of division when B is a negative value, precision of the intermediate result of the multiplication and division follows. However, if the absolute value of a literal (integer) exceeds 30, the rules of "When the Exponent is a Variable" (below) are applied.

- When B is a positive value

```
A  * A -> T1
T1 * A -> T2
T2 * A -> T3
     :
Tm * A -> Tn
```

Multiplication is repeated n times.

(n=B-1, m=n-1)

- When B is a negative value

```
A  * A -> T1
T1 * A -> T2
T2 * A -> T3
     :
Tm * A -> Tn
1/Tn
```

Multiplication is repeated n times.

(n=absolute value of B-1, m=n-1)

When the Exponent is a Variable

When the exponent is a variable (integer), the number of all digits in the intermediate result of the exponent is 30 and the number of digits in the decimal-part is the reference number of digits in the decimal-part. The reference number of digits in the decimal-part used is the value described in Appendix D.

D.3.2 Intermediate Result of the Exponent Having Floating-point Attribute

Precision of the intermediate result of the exponent having the floating-point attribute is long precision.

D.4 Attributes and Accuracy of Function Values

The following table lists the attribute and precision of numeric and integer functions.

Function Name	Function Type	Attribute and Precision of Function Value
ACOS	Numeric	Long precision floating-point
ANNUITY	Numeric	Follow the intermediate result in the expression which determines the function
ASIN	Numeric	Long precision floating-point
ATAN	Numeric	Long precision floating-point
COS	Numeric	Long precision floating-point
DATE-OF-INTEGER	Integer	8-digit fixed point

Function Name	Function Type	Attribute and Precision of Function Value
DAY-OF-INTEGER	Integer	7-digit fixed point
FACTORIAL	Integer	Follow the intermediate result in the expression which determines the function
INTEGER	Integer	Fixed point with number of digits in the integer-part of argument plus 1
INTEGER-OF-DATE	Integer	8-digit fixed point
INTEGER-OF-DAY	Integer	7-digit fixed point
INTEGER-PART	Integer	Fixed point with number of digits in the integer-part of argument
LENG	Integer	10-digit fixed point
LENGTH	Integer	Fixed point with number of digits accommodating the length of argument
LOG	Numeric	Long precision floating-point
LOG10	Numeric	Long precision floating-point
MAX	Depending on the type of argument	Follow the intermediate result in the expression which determines the function
MEAN	Numeric	Follow the intermediate result in the expression which determines the function
MEDIAN	Numeric	Follow the intermediate result in the expression which determines the function
MIDRANGE	Numeric	Follow the intermediate result in the expression which determines the function
MIN	Depending on the type of argument	Follow the intermediate result in the expression which determines the function
MOD	Integer	Follow the intermediate result in the expression which determines the function
NUMVAL	Numeric	Fixed point with the number of digits of argument
NUMVAL-C	Numeric	Fixed point with the number of digits of argument
ORD	Integer	3-digit fixed point
ORD-MAX	Integer	9-digit fixed point
ORD-MIN	Integer	9-digit fixed point
PRESENT-VALUE	Numeric	Long precision floating-point
RANDOM	Numeric	Follow the intermediate result in the expression which determines the function
RANGE	Depending on the type of argument	Follow the intermediate result in the expression which determines the function
REM	Numeric	Follow the intermediate result in the expression which determines the function
SIN	Numeric	Long precision floating-point
SQRT	Numeric	Long precision floating-point
STANDARD-DEVIATION	Numeric	Long precision floating-point
STORED-CHAR-LENGTH	Integer	Fixed-point numerals having arguments with a specific number or smaller digits

Function Name	Function Type	Attribute and Precision of Function Value
		Unique to [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET].
SUM	Depending on the type of argument	Follow the intermediate result in the expression which determines the function
TAN	Numeric	Long precision floating-point
VARIANCE	Numeric	Follow the intermediate result in the expression which determines the function

Appendix E Functional Differences

This appendix specifies the functional differences for each system.

E.1 Reference Format for Free Coding Format

References to "free coding format" are specific to the system that is attaching * in the table below.

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*		*	*	*	*	*	*

E.2 Sequential Files

PRINTER-n can be specified in the ASSIGN clause to the system that is attaching * in the table below.

For further details see the topic titled "ASSIGN Clause (Sequential File, Relative File, Indexed File)" of the "Input-Output Section" section of Chapter 4, "Environment Division".

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*	*	*	*	*	*	*	*

E.3 Data Item Definition

Handling of Numeric Values

- The system that is attaching * in the table below handles COMP-5 differently from other systems.

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
			*	*	*	*	*	*	*

- The system that is attaching * in the table below supports BINARY-CHAR/SHORT/LONG/DOUBLE data items.

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*		*	*	*	*	*	*

- The system that is attaching * in the table below extends the total number of digits in fixed-point literals to 19 (other systems have a maximum total in the integer and decimal parts of 18 digits).

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*		*	*	*	*	*	*

For further details on the data types in items 1 and 2 above, see the topic "USAGE Clause" of the "Data Description Entry" section of Chapter 5, "Data Division".

Type

The type is a function specific to the system that is attaching * in the table below.

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*		*	*	*	*	*	*

E.4 Fixed-point literals

The number of digits for fixed-point literals depends on the systems.

For more details, refer to "Fixed-point Literal" in "Chapter 1 General Rules".

E.5 Presentation Files

Destination

Allowable destinations depend on the system.

For further details see the topic titled "SYMBOLIC DESTINATION Clause (Presentation file)" of the "Input-Output Section" section of Chapter 4, "Environment Division".

Combining Specifiable Clauses and Destinations

Allowable combinations of destinations and clauses depend on the system.

For further details, see the presentation file module table and FILE-CONTROL entry in the topic titled "File-Control Paragraph" of the "Input-Output Section" section of Chapter 4, the "Environment Division".

USE FOR DEAD-LOCK Statement

The USE FOR DEAD-LOCK statement can only be specified in [UXD/DS].

SPECIAL REGISTER

EDIT-OPTION2 and EDIT-OPTION3 are specific to the system that is attaching * in the table below.

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*		*	*	*	*	*	*

E.6 Inter-Program Communication

WITH phrase

In the system that is attaching * in the table below, declare the WITH phrase in the PROCEDURE DIVISION header, CALL or ENTRY statements to specify the linkage type.

For further details see Chapter 6, the "Procedure Division", the topics "CALL Statement (Inter-Program Communication)" and the "ENTRY Statement (Inter-Program Communication)" of the "Statements" section of Chapter 6, the "Procedure Division".

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
			*	*					

National language Program name

In the system that is attaching * in the table below, a national data item or a national language nonnumeric literal can be specified in CALL and CANCEL statements.

For further details see the topics titled "CALL Statement (Inter-Program Communication)" and "CANCEL Statement (Inter-Program Communication)" of the "Statements" section of Chapter 6, the "Procedure Division".

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*		*	*	*	*	*	*

RETURNING phrase

In the system that is attaching * in the table below, declare the WITH phrase in the procedure division header PROCEDURE DIVISION and CALL statement to specify the RETURNING phrase.

For further details see the chapter titled "Procedure Division" and the topic "CALL Statement (Inter-Program Communication)" of the "Statements" section of Chapter 6, the "Procedure Division".

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*		*	*	*	*	*	*

External Name

The system that is attaching * in the table below permits an external name to be defined.

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*		*	*	*	*	*	*

E.7 Statements

WRITE Statement (Sequential File)

In the system that is attaching * in the table below, the ADVANCING phrase can be specified in the WRITE statement for print files or line sequential files. However, the ADVANCING phrase can be specified only in the WRITE statement for a print file in other system that is attaching ** in the table below.

For further details, see the topic titled "WRITE Statement (Sequential File)" of the "Statements" section of Chapter 6, the "Procedure Division".

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
**	*	*	**	*	*	**	*	*	*

USE FOR DEAD-LOCK Statement

The USE FOR DEAD-LOCK statement is specific to the system that is attaching * in the table below.

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*		*	*	*	*	*	

E.8 Functions

The following functions are specific to the system that is attaching * in the table below.

Function	DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
STORED-CHAR-LENGTH			*		*	*	*		*	*
UCS2-OF			*		*	*	*		*	*
UTF8-OF			*		*	*	*		*	*
DISPLAY-OF			*		*	*			*	*
NATIONAL-OF			*		*	*			*	*
ACP-OF										*
ENUM-AND										*
ENUM-NOT										*
ENUM-OR										*
UNICODE-OF										*

E.9 Database

The database function is native to the system that is attaching * in the table below.

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
			*	*	*	*			*

The stored procedure of the communication database function is specific to the system that is attaching * in the table below.

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
				*	*	*			*

E.10 Communication Database

Linkage using PowerAIM as a communication database function is native to [Win16].

E.11 Micro Focus Native Function

The named literal, hexadecimal literals (H"XXXX...") and the CALL statement of the Micro Focus unique function are specific to the system that is attaching * in the table below.

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*		*	*	*	*	*	*

E.12 Object-oriented Programming

The object-oriented programming function is specific to the system that is attaching * in the table below.

- Definition of object reference data item of local class
- Nesting of property or in-line invocation
- RAISING phrase in PROCEDURE DIVISION header
- Property specifier

DS	HP	Solaris	Win16	Win32	Winx64	Linux	LinuxIPF	Linux64	.NET
		*		*	*	*	*	*	

E.13 .NET Programming Function

The .NET programming function is specific to [.NET].

E.14 Quantitative System Limits

In [Win16], part of the quantitative system limits in the DATA DIVISION differs from other systems.

For further details, see the section titled "Data Division" in Appendix B, "System Quantitative Limits".

F

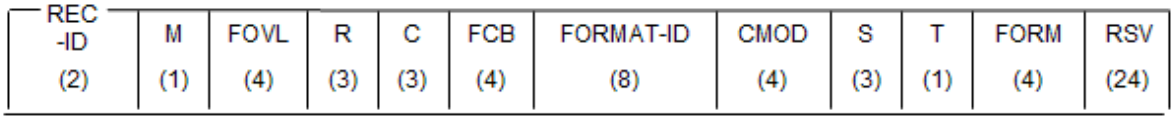
Appendix F Control Record Formats

This appendix explains the I and S control records. Certain fields are functionally meaningless. Note that some areas of the control records described here cannot be used depending on the operating system used. Refer to the "NetCOBOL User's Guide" for the details of functional scope of control records.

F.1 I Control Record

The figures below show the format of the I control record.

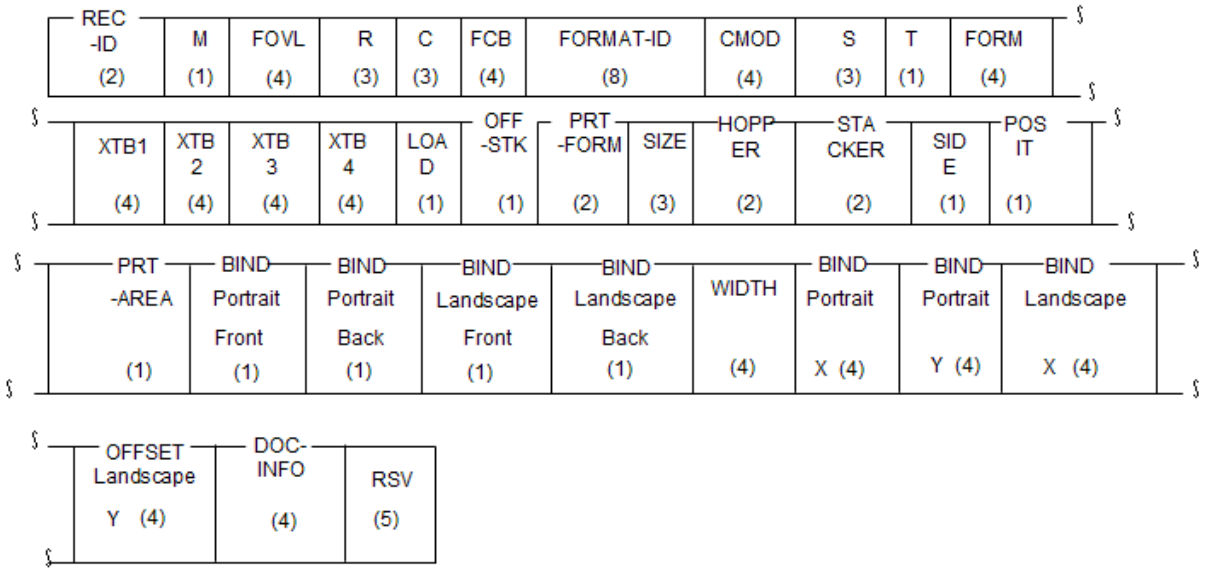
Format 1



Information

The number enclosed in parentheses in each field indicates the number of characters (bytes).

Format 2



Information

The number enclosed in parentheses in each field indicates the number of characters (bytes).

The data attributes and meanings of the I control record fields are as follows:

REC-ID: control record ID; PICTURE X (2)

The control record identifier must specify I1 for an I control record.

M: format; PICTURE X

A 0 indicates an I control record in Format-1.

A 1 indicates an I control record in Format-2.

FOVL: form overlay module name; PICTURE X (4)

Specifies the form module overlay name to use. If this field is blank, the name of the overlay file to use is specified in printer information file.

R: form overlay print count; PICTURE 9 (3)

Specifies the number of page copies subject to form overlay printing specified by FOVL. The R range is form 0 to 255. If FOVL specifies 0, the print count takes the same value as C (number of copies), and prints all copies. If the FOVL field is left blank, this field is meaningless.

C: number of copies; PICTURE 9 (3)

Specifies the number of copies of each page. The C range is from 0 to 255. If 0 is specified, the file open specification takes effect.

FCB/LPCI: FCB name / LPCI name; PICTURE X (4)

Specifies the FCB/LPCI name applied to the page. If the field is blank, the file open specification takes effect. A program cannot specify an FCB/LPCI name concurrently with a format definition.

FORMAT-ID: format definition name; PICTURE X (8)

Specifies the format definition name applied to the page, making the page a fixed-format page. If the field is blank, a non-fixed-format page results. A program cannot specify a format definition name and an FCB/LPCI name concurrently.

CMOD: copy modification module name; PICTURE (4)

Specifies the copy modification module name applied to the page. If the CMOD field is blank, the file OPEN specification takes effect.

S: copy modification start number; PICTURE 9 (3)

Specifies the segment number in the copy modification module to start copied data modification. The range is from 1 to 255. If the CMOD field is left blank, the S field is meaningless.

T: copy modification character array table number; PICTURE 9

Specifies the character array table number to be used. Use numbers 0 to 3, corresponding XTB1 to 4. If CMOD is blank, the T field is meaningless. The file open specification takes effect.

FORM: form identifier name; PICTURE X(4)

If the printer requires forms changes, you can specify the form identifier name with any name desired. If the field is blank, the program defaults to the file open forms already in use.

XTB1 through XTB4: character array table or additional character set; PICTURE X (4)

Specifies a character array table or additional character set identifier name. If this field is blank, the file open specification takes effect. Character array table specification for a printer file using a specified FORMAT clause is not available.

LOAD: dynamic load; PICTURE X

Specifies dynamic load is performed if unprintable characters are detected during printout. If field D is assigned, the program downloads the characters dynamically. If the field is blank, the file open specification takes effect.

OFF-STK: offset stack; PICTURE X

Specifies offset stacking. An O specifies offset stacking. A blank specifies no offset stacking. Offset stacking refers to the physical offsetting of the printed output, in whatever units are desired, on output from the stacker. Offset stacking must not be specified on the back surface in two-sided printing.

PRT-FORM: print format; PICTURE X (2)

Specifies the print format. The formats are: P(portrait mode), L(landscape mode), PZ(condensed print portrait mode), LZ(condensed print landscape mode), and LP(line printer mode). If the field is blank, SIZE must also be blank, and the file open specification takes effect.

SIZE: paper size; PICTURE X (3)

Specifies the paper size. The following sizes can be specified: A3, A4, A5, B4, B5, and LTR. Under [Win32] and [Winx64], a character string consisting of up to three desired characters can also be specified.

If this field is blank, PRT-FORM must also be blank, and the file open specification takes effect. For condensed printing, paper size A3 must not be specified. For line printer mode, paper size A4 must be specified. The table below shows the allowable SIZE and PRT-FORM combinations.

PRT-FORM	SIZE					
	A3	A4	A5	B4	B5	LTR
L	o	o	o	o	o	o
P	o	o	o	o	o	o
LZ	-	o	o	o	o	o
PZ	-	o	o	o	o	o
LP	-	o	-	-	-	o

o : Legal combination
- : Illegal combination

HOPPER: paper supply hopper; PICTURE X(2)

Specifies the paper supply to use.

[Win32][Winx64] The following values can be specified: P1 (primary supply hopper 1), P2 (primary supply hopper 2), or S (secondary supply hopper). If P is specified, any hopper may supply paper. If the field is blank, the file open specification takes effect. When the HOPPER field is specified, PRT-FORM and SIZE also must be specified.

[Solaris][Linux][LinuxIPF][Linux64] The following values can be specified: P1 (primary supply hopper 1), P2 (primary supply hopper 2), P3 (primary supply hopper 3), P4 (primary supply hopper 4), or S (secondary supply hopper). If P is specified, any hopper may supply paper. If the field is blank, the file open specification takes effect. When the HOPPER field is specified, PRT-FORM and SIZE also must be specified.

STACKER: paper ejection port; PICTURE X(2)

Specifies the paper ejection port for printer output. The ports are P1 (primary ejection port 1), P2 (primary ejection port 2), or S (secondary ejection port). If P is specified, any port can eject paper. If the STACKER field is blank, the file open specification takes effect.

SIDE: print surface specification; PICTURE X

Specifies the surface to be printed. An F specifies single-sided printing. A B specifies two-sided printing. If the SIDE field is blank, the file open specification takes effect.

POSIT: print position; PICTURE X

Specifies whether to position the printing surface on the front surface or back surface for two-sided printing. An F specifies printing on the front surface. A B specifies printing on the back surface. If the POSIT field is blank, the file open specification

takes effect. If SIDE is blank, the POSIT field also must be blank. If the SIDE field specifies single-sided printing, the printing must not be on the back surface.

PRT-AREA: print prohibit area; PICTURE X

Specifies prohibited printing area. An L allows printing. An N prohibits printing. If the PRT-AREA field is blank, the file open specification takes effect.

BIND: binding direction; PICTURE X

Specifies the desired direction to bind continuous multipage output. The binding format consists of four characters that specify a print format and printed sides combination. The four available fields are, in sequence, portrait mode front, portrait mode back, landscape mode front, and landscape mode back. The binding directions and characters specified for binding are L (bind left), R (bind right), U (bind up), D (bind down). The binding direction is only for multipage output. Therefore, the binding direction for the preceding page remains in effect if the BIND field is blank. The possible binding combinations are the following:

LLUU, LLUD, LLDU, LLDD,
LRUU, LRUD, LRDU, LRDD,
RLUU, RLUD, RLDU, RLDD,
RRUU, RRUD, RRDU, RRDD,
UULL, UULR, UURL, UURR,
UDLL, UDLR, UDRL, UDRR,
DULL, DULR, DURL, DURR,
DDLL, DDLR, DDRL, DDRR

WIDTH: binding width; PICTURE 9(4)

Specifies the binding width. The range is 0 to 9999, in units of 1/1440inch. If 9999 is specified, 9999 must be specified in all the OFFSET fields. If the WIDTH field is blank, all the OFFSET fields must be blank. The file open specification then starts.

OFFSET: printing origin position; PICTURE 9(4)

Specifies the printing origin position. The range is 0 to 9999 in units of 1/1440 inch. If the printing format is only portrait or landscape mode, blanks are specified in the remaining printing format fields. If 9999 is specified in all the OFFSET fields, the WIDTH field must also specify 9999. If all the OFFSET fields are blank, the WIDTH field must also be blank. The file open specification then takes effect.

DOC-INFO: Identification of Document Names; PICTURE X(4)

This specifies a string that identifies a document name. You can specify four or fewer alphanumeric characters for this field. If this field is blank, the value specified when the file is opened will be used. If a string of four or fewer alphanumeric characters is specified, that string will be associated with the corresponding document at run time.

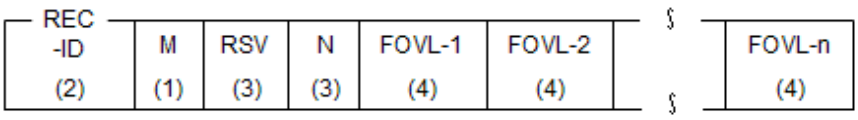
DOC-INFO is a function specific to [Win32] and [Winx64].

RSV:
used by [Win32] and [Winx64]: PICTURE X (24) or X (5)
used by [Solaris], [Linux], [LinuxIPF] and [Linux64]: PICTURE X (24) or X (9)

Must be blank.

F.2 S Control Records

The format for the S control record is shown below.



Information

The number enclosed in parentheses in each field indicates the number of character (bytes).

The data attributes and the S control record field meanings are as follows:

REC-ID: control record ID; PICTURE X(2)

Specifies S1 to indicate an S control record.

M: used by the system; PICTURE X

Must be 0.

RSV: used by the system; PICTURE X(3)

Must be blank.

N: overlay sequence count; PICTURE 9(3)

Specifies the number of form overlay module names FOVL-n.

FOVL-n: the name of a module in the form overlay sequence group; PICTURE X(4)

Specifies the number of form overlay module names specified in n. Form overlay printing is performed in the assigned order for the number of copies specified in the I control record C field.

The S control record output does not modify page attributes except the form overlay module names and form overlay printing counts. Page attributes are the number of copies and the format definition name.

Appendix G Defining a Data Item Using a Type

Generally, various clauses including PICTURE are specified to define attributes of a COBOL data item. If the data item with same attribute is necessary, the type is declared by TYPEDEF clause, and the data item can be defined referring to this type.

This appendix describes how to declare and use a type.

Note

Type is a [Win32], [Winx64], [Solaris], [Linux], [LinuxIPF], [Linux64] and [.NET] specific function.

G.1 Type

Specify the TYPEDEF clause in the data description entry of level number 01 to declare a type. The item name of the data description entry specifying the TYPEDEF clause is regarded as a type-name and used to reference a type.

The following sections give typical examples of the type declaration.

G.1.1 Type of an elementary item

In the following example, 4-digit (2-byte) and 9-digit (4-byte) binary data items are declared as types SHORT and LONG, respectively. Data items "QUANTITY" and "PRICE" are defined by referencing these types (later, a data item defined by referencing a type is called the typed item).

Example

```
 DATA  DIVISION.
 WORKING-STORAGE SECTION.
*>  Type Declaration
   01  SHORT          TYPEDEF  PIC S9(4)  COMP-5.
   01  LONG           TYPEDEF  PIC S9(9)  COMP-5.
*>  Definition of data items by referencing types
   01  QUANTITY       TYPE  SHORT.
   01  PRICE          TYPE  LONG.
*>      :
 PROCEDURE  DIVISION.
*>           :
     MOVE  0  TO  QUANTITY
     MOVE  0  TO  PRICE
```

The typed item can be referenced in the DATA DIVISION and PROCEDURE DIVISION in the same way as the normal data item unless explicitly inhibited.

"QUANTITY" and "PRICE" in the above example are the same as those defined below.

```
01  QUANTITY        PIC S9(4)  COMP-5.
01  PRICE           PIC S9(9)  COMP-5.
```

Type-names SHORT and LONG are used to identify each type declaration. They do not have any storage area. The storage area is assigned by referencing types using the TYPE clause.

The following clauses can be specified in a type:

- BLANK WHEN ZERO Clause
- JUSTIFIED Clause
- OCCURS Clause

- PICTURE Clause
- SIGN Clause
- SYNCHRONIZED Clause
- USAGE Clause

The attributes determined by these clauses are to be protected when defining a typed data item. Therefore, if an attribute of the group item to which the typed item subordinate is incompatible with the attribute contained in the type, an error occurs.

```
01 NUMDG4 TYPEDEF  PIC 9(4) COMP-5.
01 NUMDG2 TYPEDEF  PIC 9(2) COMP-5.
01 DATE-DATA  USAGE  DISPLAY.
   02  YEAR-DATA     TYPE  NUMDG4.      *>-- error
   02  MONTH-DATA    TYPE  NUMDG2.      *>-- error
   02  DAY-DATA      TYPE  NUMDG2.      *>-- error
```

G.1.2 Type of a group item

The type is used to define a group item having a specific record structure as well as an elementary item.

In the following example, "STOCKS" which is a group item with any subordinate items is declared as a type. This type is referenced to define three data items "WAREHOUSEA," "WAREHOUSEB," and "STOCK-REC".

Example

[DATA DIVISION]

```
  DATA  DIVISION.
   WORKING-STORAGE  SECTION.
*>  Type Declaration
   01  STOCKS  TYPEDEF.
      02  PRODUCT-CODE      PIC S9(4).
      02  PRODUCT-NAME.
         03  SPELL-LEN         PIC S9(4) BINARY.
         03  SPELL             PIC X(40).
     02   PRICE              PIC S9(9)  BINARY.
     02   QUANTITY           PIC S9(4)  BINARY.
*>  Definition of data items by referencing types
   01 WAREHOUSEA         TYPE  STOCKS.
   01 WAREHOUSEB         TYPE  STOCKS.
   01 STOCK-REC.
       02 WAREHOUSE-NO      PIC 9(2).
       02 STOCK-INFO     TYPE  STOCKS.
```

[PROCEDURE DIVISION]

```
*>     :
       IF  QUANTITY OF WAREHOUSEA > QUANTITY OF  WAREHOUSEB  THEN   *>---[1]
          MOVE  1  TO  WAREHOUSE-NO
          MOVE  WAREHOUSEA  TO  STOCK-INFO
       ELSE
          MOVE  2  TO  WAREHOUSE-NO
          MOVE  WAREHOUSEB  TO  STOCK-INFO
       END-IF
```

The typed item can be referenced in the same way as when it is declared as an elementary item even when the type to be referenced is a group item. When referencing the position of item subordinate to the type, it must be qualified by the name of group item which is defined by referencing the type (see [1] above).

A data item can be defined by referencing a type defined as a group item. In this case, the data item is a group item containing a subordinate item of the same name, description, and hierarchy as the item that subordinates to the data description entry of the type declaration (this group item is called the typed group item). To completely guarantee the structure of the item that subordinates to the data description entry of the type declaration, the level-number is adjusted and a slack byte is inserted to locate the item specified TYPE clause in the 8-byte boundary (see [2] below).

"WAREHOUSEA","WAREHOUSEB" and "STOCK-REC" in the above example are the same as those defined below.

```
 01  WAREHOUSEA.
     02 PRODUCT-CODE    PIC  S9(4).
     02 PRODUCT-NAME.
         03 SPELL-LEN       PIC  S9(4).
         03 SPELL           PIC  X(40).
     02  PRICE          PIC  S9(9) BINARY.
     02  QUANTITY       PIC  S9(4) BINARY.
  01 WAREHOUSEB.
     02 PRODUCT-CODE    PIC  S9(4).
     02 PRODUCT-NAME.
         03 SPELL-LEN       PIC  S9(4).
         03 SPELL           PIC  X(40).
     02  PRICE          PIC  S9(9) BINARY.
     02  QUANTITY       PIC  S9(4) BINARY.
 01  STOCK-REC.
     02 WAREHOUSE-NO    PIC  9(2).
     02 STOCK-INFO.                                  *>---[2]
         03 PRODUCT-CODE    PIC  S9(4).
         03 PRODUCT-NAME.
             04 SPELL-LEN       PIC  S9(4).
             04 SPELL           PIC  X(40).
         03 PRICE           PIC  S9(9) BINARY.
         03  QUANTITY       PIC  S9(4) BINARY.
```

G.2 Strong Type

The class of item determines the move and comparison rules for a COBOL data item. An operation not permitted for the class results in an error at compilation. The following example shows this case.

Example

```
  DATA DIVISION.
  WORKING-STORAGE SECTION.
*>  Type Declaration
   01 NUMERIC-ITEM    PIC S9(4) .
*>      :
  PROCEDURE DIVISION.
*>      :
*>  Incorrect operation
      MOVE  "ABCD" TO NUMERIC-ITEM
```

The type is simply a template of the data definition having the same structure. Even the typed item can be handled as a normal data item.

The class of a typed group item is an alphanumeric character as well as for a normal group item. In the following example, an incorrect value is set in subordinate item "SPELL-LEN" with group item move operation.

Example

```
  DATA  DIVISION.
  WORKING-STORAGE  SECTION.
```

```
*>  Type Declaration
   01 NAME-DATA TYPEDEF.
     02 SPELL-LEN    PIC S9(4)  BINARY.
     02 SPELL        PIC X(40) .
*>  Defining Data Item
   01 PRODUCT-NAME  TYPE NAME-DATA.
*>       :
  PROCEDURE  DIVISION.
*>        :
*>  Incorrect operation
      MOVE  SPACE   TO  PRODUCT-NAME
```

If a group item is defined by referencing a strong type (strongly typed group item), such an incorrect operation is inhibited. The class of a strongly typed group item is a specified type-name instead of an alphanumeric character. Therefore, if "NAME-DATA" is a strong type, operation [3] in the above example results in an error at compilation.

Declare a strong type by specifying STRONG phrase in the TYPEDEF clause of the type declaration. STRONG phrase can be specified only in the type defined as a group item. In [.NET], STRONG phrase cannot be specified.

Example

```
  DATA DIVISION.
   WORKING-STORAGE SECTION.
*>  Type Declaration
   01 NAME-DATA TYPEDEF STRONG.
     02 SPELL-LEN    PIC S9(4)  BINARY.
     02 SPELL        PIC X(40) .
```

Operations specific to the strongly typed item

The strongly typed group item differs from the normally typed group item in the following points:

- Has a native class.
- The validity of values of all subordinate elementary items is guaranteed.
- Has attributes of a type to be referenced and all subordinate items contained in the type.

Due to these differences, there are several restrictions in declaring a strong type or referencing a strongly typed group item. For example, to set a value in a strongly typed group item with group item move operation, the move on the sending side must be a strongly typed group item that references the same type.

Example

```
  DATA  DIVISION.
  WORKING-STORAGE  SECTION.
*>  Type Declaration
   01 NAME-DATA TYPEDEF STRONG.
     02 SPELL-LEN    PIC S9(4)  BINARY.
     02 SPELL        PIC X(40) .
*>  Defining data item
   01 NAMEA   TYPE NAME-DATA.
   01 NAMEB   TYPE NAME-DATA.
   01 NAMEC.
     02 SPELL-LEN    PIC S9(4)  BINARY.
     02 SPELL        PIC X(40) .
  PROCEDURE  DIVISION.
*>      :
      MOVE  NAMEA   TO  NAMEB.
```

```
      MOVE  NAMEA   TO  NAMEC.
      MOVE  NAMEC   TO  NAMEA.     *>---[4]
```

In the above example, operation [4] results in a compilation error because a normal group item is moved to a strongly typed group item.

The strongly typed group item can be used in the following functions:

- Display
- Move
- Compare
- Formal parameter or actual argument of a method

Some operations are executed only for a strongly typed item.

Normally, a group item containing an object reference item cannot be moved or compared as a group item. However, if this item is declared as a strong type, it can be moved or compared because the strongly typed group item has attribute information of all subordinate items contained in the type to be referenced.

Example

```
*>  :
  REPOSITORY.
     CLASS CONTACT.
  DATA  DIVISION.
  WORKING-STORAGE  SECTION.
*>  Declaring Strong Type
   01 CONTACT-INFO TYPEDEF STRONG.
     02 CONTACT-NUM    PIC S9(4)  BINARY.
     02 CONTACT-OBJ  OBJECT REFERENCE CONTACT OCCURS 9999.
*>     :
*>  Defining a strongly typed item
   01 TODAY-INFO           TYPE CONTACT-INFO.
   01 YESTERDAY-INFO  TYPE CONTACT-INFO.
*>     :
  PROCEDURE  DIVISION.
*>      :
      MOVE  TODAY-INFO   TO  YESTERDAY-INFO.
```

A group item containing an object reference item can be used in the following functions only when it is defined as a strongly typed item:

- Move
- Compare
- Formal parameter or actual argument of a method

When a group item is specified in a formal parameter or actual argument of a method, a normal group item is regarded as an alphanumeric data item having the same length for conformance checking. For a strongly typed group item, this conformance checking is stricter.

When a strongly typed group item is specified in a formal parameter or actual argument of a method, the item specified in the formal parameter and that specified in the actual argument must reference the equivalent type.

G.3 Type Referencing Other Types

A type can be referenced not only for defining a normal data item but also by a data description entry for the type declaration. Therefore, the "STOCKS" type described above can also be defined as shown below.

Example

[DATA DIVISION]

```
  DATA  DIVISION.
  WORKING-STORAGE  SECTION.
*>  Type Declaration
   01 SHORT  TYPEDEF   PIC S9(4)  COMP-5.
   01 LONG   TYPEDEF   PIC S9(8)  COMP-5.
   01 STOCKS  TYPEDEF.
     02 PRODUCT-CODE     PIC 9(9)  DISPLAY.
     02 PRODUCT-NAME.
        03 SPELL-LEN       TYPE SHORT.
        03 SPELL           PIC X(40).
     02 PRICE    TYPE LONG.
     02 QUANTITY  TYPE SHORT.
*>       :
*>  Defining Data item by referencing type
   01 WAREHOUSEA   TYPE STOCKS.
   01 WAREHOUSEB   TYPE STOCKS.
   01 STOCK-REC.
     02 WAREHOUSE-NUM    PIC 9(2).
     02 STOCK-INFO       TYPE STOCKS.
*>     :
```

[PROCEDURE DIVISION]

```
*>   :
     IF QUANTITY OF WAREHOUSEA > QUANTITY OF WAREHOUSEB THEN
        MOVE  1 TO  WAREHOUSE-NUM
        MOVE WAREHOUSEA  TO STOCK-INFO
     ELSE
        MOVE  2 TO  WAREHOUSE-NUM
        MOVE WAREHOUSEB  TO STOCK-INFO
     END-IF.
```

In this example, "WAREHOUSEA","WAREHOUSEB" and "STOCK-REC" have the same results as those in an example of the description that does not use the type shown in the section "Type of a group item."

If the type referenced by the type declaration is strong, note the following points:

- The type declaration containing a strongly typed item must declare a strong type.
- Information of a type referenced by the strongly typed item contained in the type declaration is stored as information of a subordinate item contained in the newly declared type.

Example

[DATA DIVISION]

```
  DATA  DIVISION.
  WORKING-STORAGE  SECTION.
*>  Type Declaration
   01 SHORT  TYPEDEF  PIC S9(4)  COMP-5.
   01 LONG   TYPEDEF  PIC S9(8)  COMP-5.
   01 NAME-DATA  TYPEDEF  STRONG.
     02 SPELL-LEN       PIC S9(4)  BINARY.
     02 SPELL           PIC X(40).
   01 STOCKS  TYPEDEF STRONG.
     02 PRODUCT-CODE    PIC 9(9) DISPLAY.
     02 PRODUCT-NAME    TYPE NAME-DATA.
     02 PRICE           TYPE LONG.
```

```
     02 QUANTITY        TYPE SHORT.
*>       :
*>  Defining Data item by referencing type
   01 WAREHOUSEA   TYPE STOCKS.
   01 STORED-PRODUCT   TYPE STOCKS.
```

[PROCEDURE DIVISION]

```
*>   :
     MOVE  PRODUCT-NAME  OF  STORED-PRODUCT  TO
                             PRODUCT-NAME  OF  WAREHOUSEA   *>---[5]
```

In the above example, "PRODUCT-NAME" which is a subordinate item of "WAREHOUSEA" is not only a subordinate item of the "STOCKS" type but also a strongly typed item with the name-type data type. Therefore, the contents of "STORED-PRODUCT" which is a strongly typed data item with the name-type data type can be set.

G.4 Effectively Using the Data Definition using a Type

To describe the following programs, it is recommended to define a data item by referencing a defined type:

- Program that references a data item having a common data structure
- Program linked with non-COBOL programs

Information

See the user's guide for details of defining a type that corresponds to a non-COBOL data type.

Index

[Special characters]

.NET
.... 560,561,562,563,564,565,566,569,581,582,583,585,588,589,590,592,593,594,598,599,606,608

[A]

ACCEPT statement....50,270,271
ACCEPT statement (command line arguments and environmental variables)....272
ACCEPT statement (nucleus)....269,271
ACCEPT statement (screen operation)....489
access mode....55
ACCESS MODE clause....54,55,111,112,281
ACCESS MODE clause (sequential, relative, indexed, and presentation files and report writer)....110
ACOS function....400
ACP-OF....654
actual argument....560
actual parameter....560
actual tables....436
ADDR function....400
ADD statement (nucleus)....273
ADVANCING PAGE phrase....145
ALL literal....15
ALL phrase....312
alphabet-name....8,89
ALPHABET clause....93,95
ALPHABETIC....612
ALPHABETIC-LOWER....612
ALPHABETIC-UPPER....612
alphabetic argument....398
alphabetic characters....1,442,473
alphabetic item....262
alphabetic screen item....204
alphanumeric argument....398
alphanumeric character....473
alphanumeric data item....167,262
alphanumeric edited data item....167,262
alphanumeric edited screen item....204
alphanumeric function....399
alternate record key....54
ALTERNATE RECORD KEY clause....54
ALTERNATE RECORD KEY clause (indexed file)....112
ALTERNATE RECORD KEY phrase....338
ALTER statement....633
ALTER statement (nucleus)....274
an EXIT PROGRAM statement....61
ANGLE phrase....101
ANNUITY function....401
applicable rule column....225
APPLY MULTICONVERSATION-MODE clause....131
APPLY MULTICONVERSATION-MODE clause (presentation file)....131
APPLY SAVED AREA clause (presentation file)....131
approximate literal....475
Area A....34
Area B....34
argument....79,398,616
ARGUMENT-NUMBER....93
ARGUMENT-VALUE....93
argument position indicator....78
arguments....78,79,398
arithmetic expression....240,241,242,398
arithmetic operators....241
arithmetic statement....264
arithmetic statements....263
ASCENDING KEY phrase....357
ASCII....682
ASIN function....401
ASSIGN clause....54,55,114
ASSIGN clause (presentation file)....115
ASSIGN clause (sequential, relative,and indexed files)....113
ASSIGN clause (sort-merge and report writer)....115
assignment compatibility....644
ATAN function....402
at end condition....58,268
AT END phrase....58,73,267
attribute....561
attribute class....561
AUTO clause....196
AUTOMATIC....57

[B]

BACKGROUND-COLOR clause....196,197
based-storage section....153
BASED ON clause....154,155
BELL clause....197
BINARY-CHAR....186,657
BINARY-SHORT/LONG/DOUBLE....657
blank line....35
BLANK LINE clause....197
BLANK SCREEN clause....197,198
BLANK WHEN ZERO clause....155,198
BLINK clause....198
BLOCK CONTAINS clause (sequential, relative, indexed files and record writer module)....140
body group....229
BOOLEAN....612
Boolean comparison....259
Boolean expression....242,243
Boolean item....263
Boolean literal....15
Boolean operators....242,452
BY CONTENT....611,635,641
BY REFERENCE....611,635,639
BY VALUE....611,635,639,640

[C]

called program....60,61,278
called programs....137
calling program....60

CALL statement..... 60,61,276,277,278,279,635
CALL statement (inter-program communication).....275,503
CANCEL statement.....64,280
CANCEL statement (inter-program communication)..... 280
CAST ALPHANUMERIC function.....402
CATCH.....652
character-strings.....4
characters.....682
character set..... 89
CHARACTERS phrase..... 312
character string literal..... 443,474
CHARACTER TYPE clause..... 155,156,157,158,159
CHAR function.....403
class..... 562
CLASS-ID paragraph..... 583
class-name..... 565
CLASS clause.....96
class condition..... 245,611
class name.....7
CLI.....561
client..... 472
closed type..... 563
CLOSE statement..... 281,282,283,461
CLOSE statement (sequential, relative, indexed and presentation files and record writer module)..... 281
CLOSE statement, dynamic..... 467
CLR..... 561,606
CLS.....561
COBOL.....1,2,4,41,658,679
COBOL data types.....606
COBOL for .NET..... 560
COBOL original type data.....561
COBOL word..... 4
code.....564
CODE-n.....94
CODE-SET clause (sequential file and report writer module)141
CODE clause..... 209
collating sequence.....89
COLLATING SEQUENCE phrase.....317,359
collection..... 561
colon..... 3,4
column name.....445,447
COLUMN NUMBER clause.....199,215,485
column specification.....447
combined condition..... 249
combined relation condition..... 254
comma..... 170
command line argument..... 77
comment entry..... 18
comment line..... 36
COMMIT statement..... 469
COMMON clause.....83
common language infrastructure..... 561
common language runtime..... 561
common language specification..... 561
common program.....83
common statement rules.....240
communication data.....472
communication database.....472,473
communication database function..... 472
comparison predicate.....449
comparisons.....613
compilation group.....576
compilation unit.....38
compile date paragraph.....83
compiler-directing sentence.....238
compiler-directing statement..... 235
complex condition..... 248,250
computer-name..... 9
computer character sets..... 2
COMPUTE statement.....44
COMPUTE statement (nucleus).....284
concatenation expression.....17
concatenation operator.....4,17
condition..... 611
condition-name data description entry..... 153
conditional sentence..... 237
conditional statement.....235
conditional variable..... 7
condition expression, complex..... 249
condition names.....7
conditions.....613
CONFIGURATION SECTION..... 592
CONSOLE IS CRT clause..... 482
constant section.....137,154
constraint..... 561
constructor..... 562
continuation line..... 36
continued line..... 36
CONTINUE statement..... 285
CONTINUE statement (nucleus).....285
CONTROL clause..... 209
CONTROL HEADING phrase.....222
CONTROL RECORDS clause (sequential file).....141
copying..... 262,263
COPY statement..... 428,429,430,431,432
CORRESPONDING phrase..... 266
COS function..... 403
COUNT IN phrase.....379
crossfooting..... 220
CRT-STATUS clause.....97
CRT STATUS clause..... 77,97
CURRENCY SIGN clause..... 98
currency symbol..... 171
CURRENT-DATE function..... 404
current receiving item.....379
current test position..... 378,379
current transfer position.....370
CURSOR clause..... 98
cursor name.....445
custom-attribute-name..... 565
CUSTOM-ATTRIBUTE clause.....593
CUSTOM-ATTRIBUTE phrase.....583,585,590,600
custom attribute..... 561

[D]
data....18,456,459,561,564,613
data-name....8,153
database....472
database function....697
database function (SQL)....436
data description entry....151,153
data division....35
Data Division
.... 135,136,137,138,139,140,141,142,143,144,146,149,150,151,154,155,161,162,163,165,176,177,178,179,180,184,191,193,196,197,198,199,200,201,202,203,205,206,207,209,210,212,215,216,217,218,220,224,225
data item....29
DATA RECORDS clause (sequential, relative, and indexed files)....142
data type mapping....606
DATE-OF-INTEGER function....404
DAY-OF-INTEGER function....405
DAY-OF-WEEK....270
DCSQL....472
DCSQLMSG....479
DCSQLSTATE....478
DCSQL statements....479
debugging line....37
DECIMAL-POINT IS COMMA clause....99,170
decimal point....170
DECLARATIVES....234,633
delegate....561,562,639
DELEGATE-ID paragraph....588
delegate-name....565
DELETE statement....286
DELETE statement (positioned)....462
DELETE statement (positioned), dynamic....468
DELETE statement (relative and indexed files)....285
DELETE statement (searched)....457
DELIMITED BY phrase....369
delimiter, pseudo-text....3
DELIMITER IN phrase....378
delimiters....378
DEPENDING ON phrase....148,149
DEPENDING phrase....46,47
DESCENDING phrase....317,359
DESTINATION clause (presentation file)....116
destination type....71
digit....473
digits....442
DISPLAY statement....50,287,288,289,290
DISPLAY statement (command line arguments and environmental variables)....289
DISPLAY statement (nucleus)....286
DISPLAY statement (screen)....492
DISPLAY statement (screen operation)....288
DIVIDE statement....44,292
DIVIDE statement (nucleus)....290
DLLIMPORT attribute....561
dynamic access mode....55
dynamic CLOSE statement....467
dynamic cursor declaration....466
dynamic DELETE statement (positioned)....468
dynamic FETCH statement....467
dynamic OPEN statement....466
dynamic SQL statement....463
dynamic UPDATE statement (positioned)....468

[E]
EBCDIC....682
EDIT-COLOR....10,28,74
EDIT-CURSOR....10,28,74
EDIT-MODE....10,28,73
EDIT-OPTION....10,28,73
EDIT-OPTION2....10,28,74
EDIT-OPTION3....10,28,74
EDIT-STATUS....10,28,74
editing sign control symbols....170
elementary item....18,260
elementary item size....169
elementary screen item....77
elementary screen items....76,193
ELSE phrase....44,302
embedded DCSQL....472,473,474,476
embedded DCSQL declare section....476
embedded DCSQL statement....476
embedded SQL....436,437,442,445,452
embedded SQL statement....436
embedded SQL statements....436,441
END-DIVIDE phrase....292
END-PERFORM phrase....330
END-READ phrase....339,345
END-STRING phrase....371,372
END-UNSTRING phrase....381,382
END-WRITE phrase....391
END DECLARATIVES....234
END KEY clause (presentation file)....116
END markers....591
end program header....82,84
ENTRY....636
ENTRY statement....293
ENTRY statement (inter-program communication)....293
enum....562
ENUM....654
ENUM-AND....655
ENUM-ID paragraph....589
enum-name....565
ENUM-NOT....656
ENUM-OR....655
ENVIRONMENT-NAME....93
ENVIRONMENT-VALUE....93
environment division....35
Environment Division....86
ENVIRONMENT DIVISION....592
environment variable....77,79
ERASE clause....199
ERASE EOL clause....200

ERASE EOS clause....200
EVALUATE statement....44,45,294,295,296
EVALUATE statement (nucleus)....294
exact numeric literal....475
exception....637
EXCEPTION-OBJECT....571
exception condition....267,268
exception conditions....633
exception object....633,637,654
EXECUTE IMMEDIATE statement....465
EXECUTE statement....464
EXIT....636
EXIT PERFORM statement....298
EXIT PROGRAM statement....61,299
EXIT PROGRAM statement (inter-program communication)299
EXIT statement (nucleus)....298
explicit scope terminator....238
extended characters....473
EXTEND phrase....387
external attribute....62
EXTERNAL clause....62,143,161,601
EXTERNAL clause (sequential, relative, indexed, and presentation files and report writer module)....143
external data record....62
external file connector....62
external medium is called its organization. There are four....53
external names....574

[F]

FACTORIAL function....406
FETCH statement, dynamic....467
field-name....565
field reference....569,573
figurative constant....11,15
file....53,67
FILE-CONTROL paragraph....103
file-identifier literal....18
file-name....8
file attribute conflict condition....58
file connector....54
file control entry....103
file description entry....137,139
file identifier name....9
file position indicator....56
file section....136
FILE SECTION....599
FILE STATUS clause....393
FILE STATUS clause (sequential, relative, indexed, presentation files and report writer)....117
FILLER....153
FINAL....222
FINALLY....652
fixed-length records....59
fixed-point numeric literal....11
fixed floating-point attribute....685
fixed format....34
fixed insertion....172
fixed insertion characters....172
floating-point attribute....685
floating-point data item....263
floating-point item....51,52
floating-point literal....12,51
floating insertion....172
floating insertion characters....172
floating point....51
FOREGROUND-COLOR clause....200,201
formal parameter....562
FORMAT clause....54
FORMAT clause (sequential and presentation files)....118
Framework....560
free format....34
FULL clause....201
function-identifier....33,398
function-name....9,11
function-name-1 clause....89
function identifier....568
function name....398
functions....654,655,656,657
function value....30,74,398

[G]

GENERATE statement....209,300
generic....562
generic class....562
generic delegate....562
generic interface....562
generic methods....581
generics....562,581
generic type....576
generic types....581
GET....602
GET PROPERTY....585
GIVING phrase....318,358,360
GLOBAL clause....61,162,601
GLOBAL clause (sequential, relative, indexed and presentation files and report writer module)....143
global name....61
GLOBAL phrase....387
GO TO statement....44,301
GO TO statement (NET)....637
GO TO statement (nucleus)....301
GRID clause....486
GROUP clause (sequential and presentation files)....118
grouped table....452
GROUP INDICATE clause....215
group items....263
group screen item....77
group screen items....76,193

[H]

header....82,84,238
hexadecimal nonnumeric literal....13,481,566
HIGHLIGHT clause....201
HIGH VALUE....15
host variable....474,477

host variable definitions....437
host variables....436,440,472,478

[I]

I-O-CONTROL paragraph....598
I-O control paragraph....130
I-O module, indexed....103
I-O status....58
I control record....700
identification division....35
IDENTIFICATION DIVISION....581
identifier....32,153,398,568,571
identifiers....569
IF statement....44,302
IF statement (nucleus)....301
imperative sentence....238
imperative statement....235
in-line comment....37
in-line method invocation....568,569
index-name....7,47,66
index comparison....260
index data item....189,304
INDEXED BY phrase....47,350
indexed file....53,103
indexed file module....137
indexed files....55
indexed I-O module....53
indexed organization file....54
INDEXED phrase....54
indicator area....34
INHERITS....583
initial attribute....63
INITIAL clause....63
INITIALIZATION statement....305
INITIALIZE statement....49,304
INITIALIZE statement (nucleus)....303
initial program....63,83
initial state of program....63
INITIATE statement....220,305
inner PERFORM statement....46
INPUT-OUTPUT SECTION....598
input-output statements....55
input item....77,204
INPUT phrase....387
INPUT PROCEDURE phrase....359
input tables....472
INSERT statement....457
INSPECT statement....49,306,308,309,310,311,313,314
instance constructor....562
INTEGER-OF-DATE function....407
INTEGER-OF-DAY function....407
INTEGER-PART function....408
integer argument....398
integer function....399
INTEGER function....406
integer item....167
integer saving item....226
inter-program communication module....10,60
interface....562
INTERFACE-ID paragraph....589
interface-name....565
intermediate result....685
intermediate results....657,685
internal attribute....62
internal file connector....62
internal names....574
internal program....38
internmediate results....657
INTO clause/USING clause....463
intrinsic data type....563
invalid key condition....58,266
INVALID KEY phrase....58,266
INVOKE....638
item controller....73

[J]

JIS 8-bit Code....682
JUSTIFIED clause....21,162,163,202

[K]

KEY IS phrase....350
key of reference....54

[L]

LABEL RECORDS clause (sequential, relative, indexed and report writer files)....143
language-name....9
LEFTLINE clause....487
left parenthesis....3
LENGTH function....409
level-number....8
library-name....8
library text....428
LINAGE-COUNTER....10,27
LINAGE clause....27,144,146
LINAGE clause (sequential file)....144
LINE-COUNTER....10,27
line-counter....81
LINE-counter....208
LINE NUMBER clause....202,216,226,487
line sequential organization file....54
LINE SEQUENTIAL phrase....54
linkage section....137
LINKAGE SECTION....599
literal....11,398,566
literal item....77
literals....17,442
local name....61
LOCK MODE clause....57,119
LOCK MODE clause (sequential, relative, and indexed files)119
LOCK MODE IS AUTOMATIC....119
LOCK MODE IS EXCLUSIVE....119
LOCK MODE IS MANUAL....120
LOG10 function....410
LOG function....410

LOW-VALUE..... 15
LOWER-CASE function..... 411
LOWLIGHT clause..... 203

[M]

main program..... 61
managed code..... 563
mantissa..... 51,189
MANUAL..... 57
markers..... 591
MAX function..... 411
MEAN function..... 412
MEDIAN function..... 412
MEMORY SIZE clause..... 87
merge..... 69
merge module..... 67
MERGE statement..... 67,314,315,316,359,360
merging..... 68
MESSAGE CLASS clause..... 120
MESSAGE CLASS clause (presentation file)..... 120
MESSAGE CODE clause (presentation file)..... 120
MESSAGE MODE clause (presentation file)..... 121
MESSAGE OWNER clause (presentation file)..... 122
MESSAGE SEQUENCE clause (presentation file)..... 122
metadata..... 563
method..... 562,563,564,568,569,636
METHOD-ID paragraph..... 585
method-name..... 565
method signature..... 563
Micro Focus..... 482
Microsoft .NET..... 560
MIDRANGE function..... 413
MIN function..... 413
mnemonic-name..... 8,89
MODE IS BLOCK clause..... 495
MOD function..... 414
modifier..... 568,570
modules..... 41
MOVE statement..... 43,260,320
MOVE statement (nucleus)..... 319
moving..... 643
MULTIPLE FILE TAPE clause (sequential file and report writer)
..... 132
MULTIPLY statement..... 44,322

[N]

names..... 574
namespace..... 563
National..... 495,496,500
national character..... 473
national character argument..... 398
national characters..... 2,442
national character string literal..... 444,475
National data item..... 167
national edited data item..... 168
national edited screen item..... 204
national function..... 400
NATIONAL function..... 405,414,415
national hexadecimal nonnumeric literal..... 14
national nonnumeric characters..... 259
national nonnumeric literal..... 14
negated condition..... 249
NEW..... 562
NEXT GROUP clause..... 217,226
next report group..... 225
no at end condition..... 268
no invalid key condition..... 267
nonnumeric comparison..... 257
nonnumeric literal..... 12
no other exception condition..... 268
no size error condition..... 265
NOT AT END-OF-PAGE phrase..... 393
NOT AT END phrase..... 267
NOT INVALID KEY phrase..... 266
NOT ON SIZE ERROR phrase..... 265
nucleus module..... 41
Nucleus module..... 44,49,50,52
NULL..... 571
NUMERIC..... 612
numeric character argument..... 398
numeric characters..... 1
numeric comparison..... 258
numeric data item..... 166,262
numeric edited data item..... 167,262
numeric edited screen item..... 204
numeric function..... 399
numeric literal..... 11,442,443
NUMVAL-C function..... 416
NUMVAL function..... 415

[O]

object..... 564
object computer paragraph..... 87
object identifier..... 568,569,571
object modifier..... 568,570
OBJECT paragraph..... 585
object property..... 568,572
OBJECT REFERENCE..... 605
object reference identifiers..... 569
object references..... 643
occurrence count..... 46
OCCURS clause..... 46,163,164,165,351,487,602
OCCURS DEPENDING ON clause..... 164
ON OVERFLOW phrase..... 385
ON SIZE ERROR..... 633
ON SIZE ERROR phrase..... 44,265
OO COBOL..... 560
open mode..... 55
OPEN statement..... 323,324,325,326,327,460
OPEN statement (presentation file)..... 326
OPEN statement (report writer)..... 327
OPEN statement (sequential, relative, and indexed files)..... 322
OPEN statement, dynamic..... 467
open type..... 563
operand combinations..... 256

operands....266
operation module....77,79
operator....569,573
optional word....10
ORD-MAX function....417
ORD-MIN function....418
ORD function....417
organization....576
ORGANIZATION clause....54
ORGANIZATION clause (indexed file)....123
ORGANIZATION clause (presentation file and report writer)123
ORGANIZATION clause (relative file)....123
ORGANIZATION clause (sequential file)....123
ORGANIZATION IS SEQUENTIAL clause....123
original type data....561
outer PERFORM statement....45
output item....77,204
OUTPUT PROCEDURE....358
OUTPUT PROCEDURE phrase....317,360
output tables....472
OVERLINE clause....487
overloading....563
OVERRIDE....585

[P]

PADDING CHARACTER clause (sequential file)....124
PAGE-COUNTER....10,27,208
page break rule....225
PAGE clause....210,211
page counter....81
page footing group....231
page heading group....222,228
paragraph....235
paragraph-name....7,27
paragraph name....235
parameter....64
parameter conformance....616
parentheses....3
PERFORM....641
PERFORM statement....45,46,330,331,332,387
PERFORM statement (nucleus)....328
PERFORM statement....46
period (separator)....238
permitted comparisons....613
PICTURE....174
PICTURE character....377
PICTURE character-string....18,166
PICTURE clause....4,18,165,166,203
PICTURE symbols....166,170
pointed....31
pointed qualifier....31
pointer data....259
pointer data function....400
pointer data item....50,263
POINTER phrase....378
pointer qualification symbol....4
pointers....50
POSITIONING UNIT clause....99,176
positioning unit name....7
possible comparions....613
predefined object identifier....568,571
predicate....449
PREFIX phrase....432
PREPARE statement....463
PRESENT-VALUE function....419
presentation file....72,105
presentation file module....10,70,105
presentation rule tables....225
prime record key....54
print file....54
printing....225
PRINTING MODE clause....99,101
PRINTING POSITION clause....176
print mode name....7
PRIVATE....585
procedure....234
Procedure....400
procedure branching statement....237
procedure division....35
Procedure Division
....234,235,238,240,241,244,245,247,248,249,255,257,258,259,
260,264,265,266,267,269,271,273,274,275,280,281,284,285,
286,288,289,290,293,294,298,299,301,304,305,306,314,319,
322,326,327,328,336,343,345,346,349,354,356,362,367,373,
375,376,385,387,388,389,397,398,400,401,402,403,404,405,
406,407,408,409,410,411,412,413,414,415,416,417,418,419,
420,421,422,423,424,425,426,503,608
PROCESSING CONTROL clause (presentation file)....124
PROCESSING MODE clause (presentation file)....124
PROCESSING TIME clause (presentation file)....125
program....38,39,63,564
PROGRAM....636
PROGRAM-ID paragraph....582
program-name....8,66
program-name literal....17
program-prototype-name....565
PROGRAM-STATUS....10,67
PROGRAM COLLATING SEQUENCE clause....88
program end statements....61
program identification number area....35
program name....82
program nesting....38
program prototype....563
program structure....576,577
PROMPT clause....488
property....568,572,585
PROPERTY....593
property-name....565
PROPERTY clause....602
PROTECTED....585
pseudo-text....428
pseudo-text delimiter....3
Pseudo-text delimiter....428
pseudo-text delimiters....4

PUBLIC....585

[Q]

qualifier....26,28
quantitative system limits....698
quotation mark....3
quotation marks....3
QUOTE....15

[R]

RAISING....637
random access mode....55
RANDOM function....419
RANGE function....420
READ statement....57,338,339,340,341,344,497,498,499
READ statement (presentation file)....343
READ statement (relative and indexed files)....496
READ statement (sequential, relative, and indexed files)....336
receiving item....378,616
record....18,57
record-name....8
record area....59
RECORD clause....147
RECORD clause (presentation file)....149
RECORD clause (report writer module)....149
RECORD clause (sequential, relative and indexed files)....146
RECORD DELIMITER clause (sequential file)....126
record description entry....138
RECORD KEY clause....54
RECORD KEY clause (indexed file)....126
RECORD KEY phrase....338
record keys....54
RECORD phrase....339
record sequential file....54
RECORD VARYING phrase....346
REDEFINES clause....177,304,433,604
reference....569,573
reference format....33,34
reference modification....29
reference modifier....568
reference type....563
relation condition....244
relation condition, combined....254
relative file....52,103
relative file module....137
relative files....55
relative I-O module....52
RELATIVE KEY phrase....54,340,361
RELATIVE phrase....54
RELEASE statement....345
RELEASE statement (sort-merge)....345
REM function....420
RENAMES clause....51,178
REPLACE....434
REPLACE OFF statement....70
REPLACE statement....70,428,433,434,435
REPLACING phrase....431
report-name....8
REPORT clause (report writer module)....150
report description entry....207
report file....80,81,110
report file module....81
report footing group....222,232
REPORT FOOTING phrase....222
report group description entry....212
report heading group....222,226
REPORT SECTION....81
report section....138
report writer module....10,79,137
REPOSITORY paragraph....594
REQUIRED clause....205
required word....9
RERUN clause (sequential, relative and indexed files)....132
RESERVE clause (sequential, relative, and indexed files and report writer)....127
reserved word....9
reserved words....565,566
RESET phrase....223
RETURN-CODE....10,67
RETURNING....632,635,639
RETURN statement....346
RETURN statement (sort-merge)....345
REVERSE-VIDEO clause....205
REVERSE function....421
REWRITE statement....347,348,349
REWRITE statement (sequential, relative, and indexed files)346
right parenthesis....3
rolling forward....220
ROUNDED phrase....44,264,292
rules....611,613,644
run unit....61

[S]

SAME clause (sequential, relative, indexed and presentation files, sort-merge, and report writer)....132
S control record....703
screen....76,77
screen data description entry....193
screen form descriptor....71,72,73
screen handling module....76
screen input status....77
screen item....76,193
screen items....76
screen section....138
SEARCH statement....48,351,352,353,354
SEARCH statement (nucleus)....349
secondary entry name....66
SECTION....235
section-name....7
SECURE clause....205,484
SELECT clause (sequential, relative, indexed files and report writer)....127
SELECT clause (sort-merge and presentation file)....127
SELECTED FUNCTION clause (presentation file)....128
SELECT statement....456

SELF....571,572
sending item....378,616
sentence....237
SEPARATE....379
separator....2,235
separator comma....2,3
separator period....2,3
separator semicolon....2
separator space....2,3
sequence number area....34
sequential access mode....55
sequential file....52,103
sequential file module....137
sequential files....55
sequential I-O module....52
sequential organization file....54
server....472
service name....474
SESSION CONTROL clause (presentation file)....128
SET....602,643,644
set function specification....448
SET PROPERTY....585
SET statement....47,355,356
SET statement (nucleus)....354
SIGN clause....179,206,217,218
sign condition....248
SIM function....423
simple insertion....171
SIN function....421
SIZE clause....488
size error condition....265
slack bit....21,23,24,25
slack byte....21,22
sort....69
SORT-CORE-SIZE....10
sort-merge file....67,68,105
sort-merge file description entry....150
sort-merge module....10,67,137
SORT-STATUS....10
SORT-STATUS....69
sorting....68
sort module....67
SORT statement....67,68,357,358
SORT statement (sort-merge)....356
SOURCE clause....218,221
source computer paragraph....86
source program text....428
source statement manipulation function....428
source text manipulation module....69
source unit....576
SPACE....15
SPECIAL-NAMES....592
special-names paragraph....482
special characters....1,442,473
special insertion....172
special names....89
special names paragraph....89
special register....10,60,67
special use words....10
SQL....436
SQL, embedded....436
SQL connection....469
SQLMSG....440
SQL statement....444,445,455
SQL statement identifier....445
SQL statements....442
SQL statements, embedded....436
SQL transaction....469
SQRT function....422
standard alignment rule....21
STANDARD DEVIATION function....422
START statement....361,362,364,365,500
START statement (indexed file)....362,500
START statement (relative file)....361,500
statement....237
statements....238,263
Statements....269
static....563
static data....564
static method....564
STATIC paragraph....584
STOP RUN statement....50,61,367
STOP statement....367,652
STOP statement (nucleus)....367
STRING statement....49,369,370,371,372
STRING statement (nucleus)....367
stucture....576
sub-program....61
subscripting....28
subtotaling....220
SUBTRACT statement....44,374
SUBTRACT statement (nucleus)....373
SUM clause....218,219,220
SUPER....571,572
SUPPRESS statement....222,375
SWITCH-0....93
SWITCH-8....93
SWITCH-n....92
switch-status condition....248
symbolic-character....7,89
Symbolic-character....15
symbolic-constant....7,89
SYMBOLIC CHARACTERS clause....101,102
SYMBOLIC CONSTANT clause....51,102
SYMBOLIC DESTINATION clause (presentation file)....129
symbols....170
SYNCHRONIZED clause....21,165,180,181
SYSOUT....288
system-name....8
System.Exception....653
System.Exception class....633

[T]

table....452

table,grouped.....452
table, truth.....449
table element.....46,47,48
table name.....445
table names.....474
tables.....436
tables, actual.....436
tables, input.....472
tables, output.....472
tables, view.....436
TALLYING IN phrase.....378
TAN function.....423
target specification.....446
TERMINATE statement.....375,376
TERMINATE statement (report writer).....375
TEST BEFORE phrase.....332
testing.....611
TEST phrase.....332
text.....428
text-name.....27
text-name literal.....18
text name.....7
text words.....428
THEN phrase.....44,302
token.....444
transaction, current.....469
transcibing.....643
truth table.....449
TRY statement.....652
type.....562,563,564
type argument.....564
TYPE clause.....220
type condition.....613
type object.....564
TYPE OF.....569,573
type parameter.....561,564,576
type parameters.....581

[U]

UNDERLINE clause.....206
UNICODE-OF.....657
uniqueness of reference.....25
unit.....576
UNIT CONTROL clause (presentation file).....130
UNLOCK statement.....57
UNLOCK statement (sequential, relative, and indexed files).....376
unmanaged code.....564
UNSTRING statement.....49,378,380,381,382,384,385
UNSTRING statement (nucleus).....376
UPDATE statement (positioned).....462,463
UPDATE statement (positioned), dynamic.....468
UPDATE statement (searched).....458
UPON phrase.....288
UPPER-CASE function.....424
USAGE clause.....184,188,206,207,224,605
USAGE IS INDEX clause.....185,189
USE AFTER EXCEPTION.....653
USE AFTER STANDARD EXCEPTION procedure.....318
USE BEFORE REPORTING.....223
USE BEFORE REPORTING procedure.....388
USE BEFORE REPORTING statement.....388
USE BEFORE REPORTING statement (creation of reports).....387
USE FOR DEAD-LOCK statement.....388,695
USE FOR DEAD-LOCK statement (presentation file).....388
USE FOR DEAD-LOCK statements.....389
user-defined word.....5
user-defined words.....25,65,564,565
USE statement.....386,387
USE statement (creation of sequential, relative, indexed, and presentation files and reports).....385
USING.....616
USING phrase.....359

[V]

VALUE clause.....51,191,192,193,207,224,225,607
value expression.....448
VALUE OF clause (sequential, relative and indexed files and report writer module).....150
value specification.....446
value type.....564
variable-length records.....59
variable format.....34
variable position in the record.....46
VARIANCE function.....425
view tables.....436
virtual tables.....436
volume indicator.....56

[W]

WHEN-COMPILED function.....426
WHEN phrase.....45
WITH DEBUGGING MODE clause.....37,87
WITH LOCK.....57
WITH LOCK ON RECORD.....119
WITH NO ADVANCING phrase.....288
WITH NO LOCK.....57
WITH NO LOCK phrase.....339
WITH NO REWIND phrase.....281
WITH phrase.....239,240,279
WITH [NO] LOCK.....57
working-storage section.....137,154
WRITE statement.....54,145,389,390,391,393,394,395,396,397
WRITE statement (presentation file).....397
WRITE statement (relative and indexed files).....394
WRITE statement (sequential file).....697

[Z]

ZERO.....15
ZERO-FILL clause.....489
zero suppression.....173
zero suppression characters.....173